NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE USE OF ELECTRICAL TRANSMISSION LINE
THEORY TO PREDICT THE PERFORMANCE
OF SPACECRAFT RADIATORS

by

Steven M. Smith

March 1992

Thesis Advisor: Allan D. Kraus

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION AVAILABILITY OF REPORT
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable) 39	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
THE USE OF ELECTRICAL TRANSMISSION LINE THEORY TO PREDICT THE PERFORMANCE OF SPACECRAFT RADIATORS (UNCLASSIFIED)

12 PERSONAL AUTHOR(S)
Smith, Steven Mark

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ____ TO ____	March 1992	137

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of the Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Thermal Transmission Matrix, Cascade Algorithm, Extended Surface, Cooling Fin, Radiative Fin, Longitudinal Fin, Rectangular Fin, Trapezoidal Fin, Triangular Fin, Optimum

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The cascade algorithm that is used for extended surface analysis depends on a new parameterization called the thermal transmission matrix to represent a single fin. This thermal transmission matrix, which is intended to replace the more familiar fin efficiency as a design and analysis parameterization, is a linear transformation that maps conditions of heat flow and temperature at the fin tip to heat flow and temperature conditions at the fin base. The cascade algorithm was derived by resorting to an analogy between a fin and the electrical transmission line. The cascade algorithm permits a fin to be subdivided into many subfins each having a thermal transmission matrix and then the individual transmission matrices for each of the subfins can be used, via a simple matrix product to form an overall equivalent thermal transmission matrix for the entire fin. This thesis develops a thermal transmission matrix for the radiating rectangular, trapezoidal, and triangular fins both for the free space and non-free space environments. Test cases have been run and their solutions exactly match those contained in the literature. The thesis concludes with optimization studies for

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Allan D. Kraus	22b TELEPHONE (Include Area Code) (408) 646-2730	22c OFFICE SYMBOL EC/Ks

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE Unclassified

S/N 0102-LF-014-6603

(Continued Block 18) Fin, and Thermal Transmission Line.

(Continued Block 19) each profile considered where it is observed that simple algebraic equations can be employed to describe the optimum geometry.

The Use of Electrical Transmission Line Theory to Predict The Performance of Spacecraft Radiators

by

Steven M. Smith
Lieutenant, United States Navy
B.S.E.E., University of Kansas, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
(SPACE SYSTEMS ENGINEERING)

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

ABSTRACT

The cascade algorithm that is used for extended surface analysis depends on a new parameterization called the thermal transmission matrix to represent a single fin. This thermal transmission matrix, which is intended to replace the more familiar fin efficiency as a design and analysis parameterization, is a linear transformation that maps conditions of heat flow and temperature at the fin tip to heat flow and temperature conditions at the fin base. The cascade algorithm was derived by resorting to an analogy between a fin and the electrical transmission line. The cascade algorithm permits a fin to be subdivided into many subfins each having a thermal transmission matrix and then the individual transmission matrices for each of the subfins can be used, via a simple matrix product to form an overall equivalent thermal transmission matrix for the entire fin. This thesis develops a thermal transmission matrix for the radiating rectangular, trapezoidal, and triangular fins both for the free space and non-free space environments. Test cases have been run and their solutions exactly match those contained in the literature. The thesis concludes with optimization studies for each profile considered where it is observed that simple algebraic equations can be employed to describe the optimum geometry.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

TABLE OF SYMBOLS

$A(x)$	Cross Sectional Area, m^2
A_p	Profile Area, m^2
A_{po}	Optimum Profile Area, m^2
b	Fin Length, m
b_o	Optimum Fin Length, m
h_{cv}	Convective Heat Transfer Coefficient, watts / m^2 K
h_r	Radiative Heat Transfer Coefficient, watts / m^2 K
k	Thermal Conductivity, watts / m K
L	Fin Depth, m
PN	Profile Number, dimensionless
q_a	Fin Tip Heat Flow, watts
q_b	Fin Base Heat Flow, watts
q_{cd}	Heat Flow By Conduction, watts
q_{cv}	Heat Flow By Convection, watts
q_r	Heat Flow By Radiation, watts
$S(x)$	Surface Area, m^2, a function of x
T	Local Temperature, K
T_a	Fin Tip Temperature, K

T_{av}	Average Temperature, K
T_b	Fin Base Temperature, K
T_s	Temperature of the Surrounding Environment, K
V	Environmental Parameter, dimensionless
Y_b	Characteristic Thermal Admittance of the Fin, watt / °C
Y_{bo}	Optimum Characteristic Thermal Admittance of the Fin, watt / °C
Y_i	Thermal Input Admittance of the Fin, watt / °C
x	Position in the X-direction, m
Z_b	Characteristic Thermal Impedance of the Fin, °C / watt
Z_{bo}	Optimum Characteristic Thermal Impedance of the Fin, °C / watt

Greek Symbols

γ	Arc Length Along the X-axis where the Edge Joins the Top and Bottom Faces, m
$\delta(x)$	Fin Width at Position x, m
δ_o	Optimum Fin Width at Position x, m
ϵ	Emissivity, dimensionless
η	Fin Efficiency, dimensionless
θ	Temperature Excess, dimensionless
ρ	Density, kg / dm^3 (1 dm^3 = 10^{-3} m^3)

T	Thermal Transmission Matrix Containing Elements A, B, C, D
	A & D, dimensionless
	B Impedance, °C / watt
	C Admittance, watt / °C
T^{-1}	Inverse Thermal Transmission Matrix Containing Elements A′, B′, C′, D′
	A′ & D′, dimensionless
	B′ Impedance, °C / watt
	C′ Admittance, watt / °C

ACKNOWLEDGEMENT

The author wishes to acknowledge Professor Kraus, for his guidance, patience, and humor throughout this endeavor. Thank you very much; your help made this all possible.

Finally I want to thank my beautiful wife, Deanna, who supported me, but more importantly, helped to maintain my sanity during the long arduous hours sitting at the computer terminal; and to my son Andrew who made everything fun.

I. INTRODUCTION

Since the advent of the space age, extended surfaces have been extensively studied as a means for heat rejection from all sorts of vehicles. Because weight optimization is of the utmost importance in spacecraft design, it is desirable to have an accurate technique that will rapidly optimize the geometry of the extended surface used for rejecting the heat absorbed on or generated within the vehicle. To this point in time, the optimization problem involved the time consuming solutions of second order non-linear differential equations, by numerical methods or by cumbersome and possibly inaccurate graphical methods. A more advantageous method for determining the minimum mass, maximum radiant heat rejection extended surface geometry has been needed.

The purpose of this study is to employ the new parameterization called the thermal transmission matrix to represent a single fin to evaluate the performance of several longitudinal fins. Each fin studied is subdivided into 100 subfins and the thermal transmission matrix is calculated for each subfin. Then, by a simple matrix multiplication of the individual subfin thermal transmission matrices, the overall transmission matrix is computed. Not only will this method produce the base and tip conditions of the overall fin entity, but the temperature and heat flow at any point between the fin base and fin tip can be determined. This iterative method considers the spacecraft radiant heat rejection problem in both the free and non-free space environments. Optimization must be considered and simple algebraic equations that describe the fin geometry are formulated for the longitudinal fins of rectangular, trapezoidal, and triangular profiles.

This new approach to fin synthesis and analysis permits an exact solution to any degree of accuracy that the user wishes to specify and produces the result in a computational efficient manner.

II. THEORETICAL PRESENTATION

A. GENERALIZED DIFFERENTIAL EQUATION

There are three distinct modes by which heat flows from one point to another. A rather simplistic definition of each can be stated. Conduction is the mode by which heat flows through a material by molecular motion. If the heat flows from a confining surface to a moving fluid, the mode is known as convection. If no intervening medium is present and the thermal energy is transported from one point to another in the form of electromagnetic waves, the mode is known as radiation or radiant heat transfer.

The flow of heat by conduction is proportional to the temperature gradient and cross sectional area perpendicular to the direction of heat flow.

$$q_{cd} \propto - A(x) \frac{dT(x)}{dx} \tag{2.1}$$

Insertion of a proportionality constant yields

$$q_{cd} = - k\, A(x) \frac{dT(x)}{dx} \tag{2.2}$$

where the minus sign assures a positive heat flow in the presence of a temperature gradient which must be negative. This serves to define the thermal conductivity of the material written as k(T) because it is a material property almost always a function

$$k(T) = \frac{q_{cd}}{-A(x)\,\frac{dT(x)}{dx}} \tag{2.3}$$

of temperature. Thus, the heat flow at x normal to a surface of area A(x) at a temperature T(x) is written as

$$q_{cd} = -\,k(T)\,A(x)\,\frac{dT(x)}{dx} \tag{2.4}$$

In convection the heat flow from a confining surface at temperature T(x) to a fluid whose bulk temperature is T_s will be proportional to the temperature difference and the amount of surface area.

$$q_{cv} \propto S(x)\,[\,T(x) - T_s\,] \tag{2.5}$$

Insertion of a proportionality constant permits the representation

$$q_{cv} = h_{cv}\,S(x)\,[\,T(x) - T_s\,] \tag{2.6}$$

and this serves to define the convective heat transfer coefficient h_{cv} as

$$h_{cv} = \frac{q_{cv}}{S(x)\,[\,T(x) - T_s\,]} \tag{2.7}$$

and thus Newton's Law of Cooling is written as

$$q_{cv} = h_{cv}(x,T)\; S(x)\; [\; T(x) - T_s \;] \tag{2.8}$$

where the convective heat transfer coefficient h_{cv} is written as a function of position and temperature.

Radiation requires no intervening medium and, for a perfect emitter and absorber in full view of one another, the familiar Stefan-Boltzmann law shows that the heat flow is proportional to the surface area and the difference of the fourth powers of the absolute temperature.

$$q_r = \sigma\; S(x)\; [\; T_1^4 - T_2^4 \;] \tag{2.9}$$

The proportionality constant used here is the Stefan-Boltzmann constant, $\sigma = 5.66961 \times 10^{-8}$ watt / m^2 K^4.

If the surfaces are not perfect emitters and not perfect absorbers and they are not in full view of one another, the departure from perfection is via the emissivity factor F_ϵ and the shape or arrangement factor F_A.

$$q_r = \sigma\; F_A\; F_\epsilon\; S(x)\; [\; T_1^4 - T_2^4 \;] \tag{2.10}$$

The problem of a finned surface radiating to outer space simplifies the foregoing equation (2.10). The temperature of the fin becomes T(x) and, because outer space is at about 2 K, T_2 can be assumed to be equal to 0. [Compare $T(x)^4 = 200^4$ to $T_s^4 = 2^4$]. Moreover, F_A is equal to 1, the value of a small body in a large enclosure (small

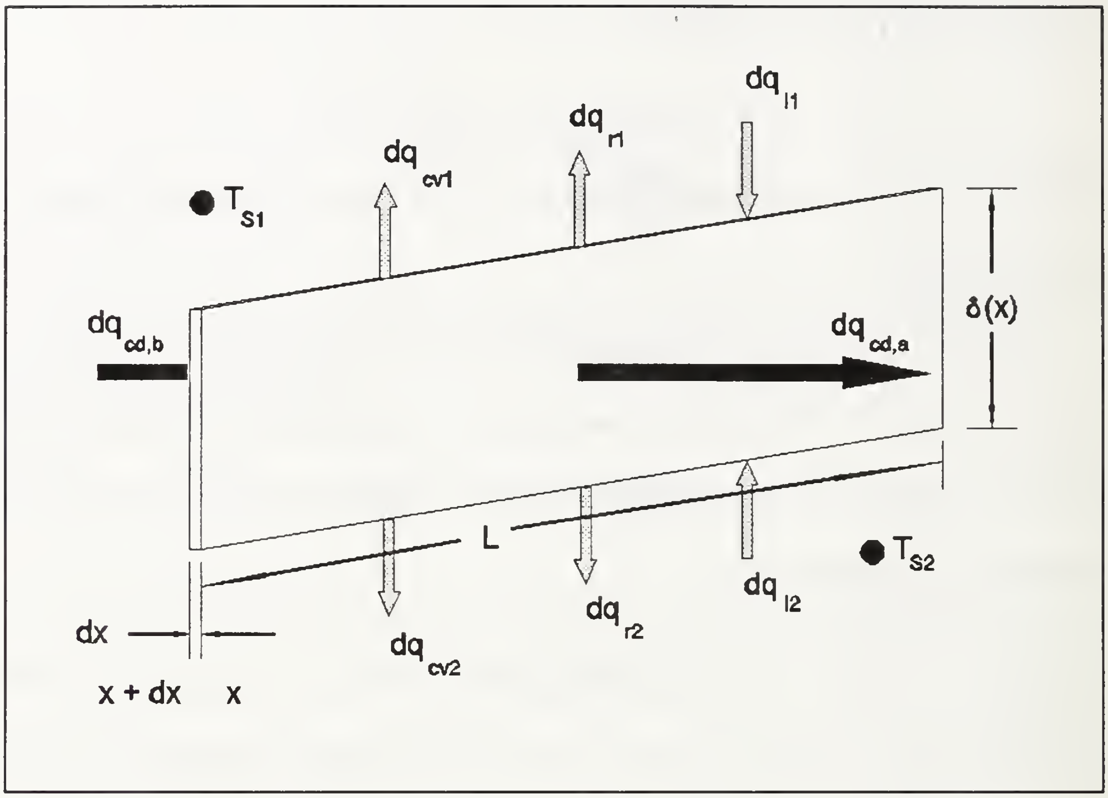

Figure 2.1 Energy Balance

fin in a large enclosure, outer space). Finally the emissivity factor for this case (small body in a large enclosure) is $F_\epsilon = \epsilon(x,T)$. The emissivity ϵ is a property of the radiating fin surface and temperature. Thus equation (2.10) is rewritten as

$$q_r = \sigma \; \varepsilon(x,T) \; S(x) \; T(x)^4 \tag{2.11}$$

The difference between the rates of heat entering and leaving the element in Figure 2.1 by conduction ($dq_{cd,b}$ - $dq_{cd,a}$) must, in the steady state, be equal to the rates of heat dissipated from the face, L dx by convection (dq_{cv1} and dq_{cv2}) and radiation (dq_{r1} and dq_{r2}) plus any rates of heat that arrive from external sources (dq_{i1}

and dq_{i2}).

The presence of two terms for dq_{cv}, dq_r, and dq_i permit unequal values on the two faces. Thus with

$$dq_{cd,a} - dq_{cd,b} = \frac{d}{dx}\left[k(x)\ \delta(x)\ L\ \frac{dT}{dx}\right] dx \tag{2.12}$$

the two convective dissipation terms are

$$dq_{cv1} + dq_{cv2} = h_{cv1}(x,T)\ L\ dx\left[\ T - T_{S1}\ \right] + h_{cv2}(x,T)\ L\ dx\left[\ T - T_{S2}\ \right] \tag{2.13}$$

and the two radiation dissipations terms with $F_A = 1$ and $F_\epsilon = \epsilon(x,T)$ are

$$dq_{r1} + dq_{r2} = \sigma\ \epsilon(x,T)\ L\ dx\left[\ T^4 - T_{S1}^4\ \right] + \sigma\ \epsilon(x,T)\ L\ dx\left[\ T^4 - T_{S2}^4\ \right] \tag{2.14}$$

The energy balance with $T_{S1} = T_{S2} = 0$ is

$$\frac{d}{dx}\left[\ k(x)\ \delta(x)\ L\ \frac{dT}{dx}\right] dx = \left[\ h_{cv1}(x,T) + h_{cv2}(x,T)\ \right] L\ dx\ T + \sigma\ \epsilon(x,T)\ L\ dx\ T^4 + q_{i1} + q_{i2} \tag{2.15}$$

In space, if either face of the fin is subjected to an external heat input of the form

$$q_i = \alpha\ L\ E\ dx \tag{2.16}$$

where E consists of external heat inputs from solar, E_S, and/or terrestrial, E_T, sources then equation (2.15), the steady state thermal energy balance for a differential element dx, is rewritten as

$$\frac{d}{dx}\left[k(x)\ \delta(x)\ L\ \frac{dT}{dx}\right] dx = \left[\ h_{cv1}(x,T) + h_{cv2}(x,T)\ \right] L\ T\ dx$$

$$+ \sigma\ \epsilon(x,T)\ L\ T^4\ dx + 2\ \alpha\ L\ E\ dx \qquad (2.17)$$

B. LIMITING ASSUMPTIONS

Further assumptions must be applied to the generalized differential equation (2.17) before a mathematical solution can be attempted. The assumptions employed by Murray[1] and Gardner[2] are used for a starting point for further analysis. The Murray-Gardner Assumptions listed in Table 1 are essential in simplifying the formulation of the generalized differential equation (2.17) so that mathematical analysis is possible. Consider an arbitrary symmetrical longitudinal fin shown in Figure 2.2 of length (b-a), depth L, and width δ(x) dissipating radiant heat to a constant temperature T_s surrounding environment.

The profile area A_p is defined as

$$A_p = \int_{x=b-a}^{x=b} \delta(x)\ dx \qquad (2.18)$$

The initial conditions at the fin tip are

$$T_a = T(\ x=a\ ) \tag{2.19}$$

and

$$q_a = q(\ x=a\ ) \tag{2.20}$$

Applying the tenth Murray-Gardner assumption (Table 1), that no heat leaves from the tip of the fin, equation (2.20) leads to

$$\frac{dT(\ x=a\ )}{dx} = 0 \tag{2.21}$$

Two more boundary conditions are taken at the fin base

$$T(\ x=b\ ) = T_b \tag{2.22}$$

and

$$q(\ x=b\ ) = q_b \tag{2.23}$$

Applying the fifth and ninth Murray-Gardner assumptions (Table 1), allows one to ignore the radiant heat flow from the edges. The surface area S(x) perpendicular to the radiant heat flow is given by

$$S(x) = 2\ L\ \gamma \tag{2.24}$$

The face area S(x) through which the heat is dissipated depends on the arc length γ, a function of the fin length b.

The cross sectional area A(x) perpendicular to the heat flow by conduction is

$$A(x) = L\ \delta(x) \tag{2.25}$$

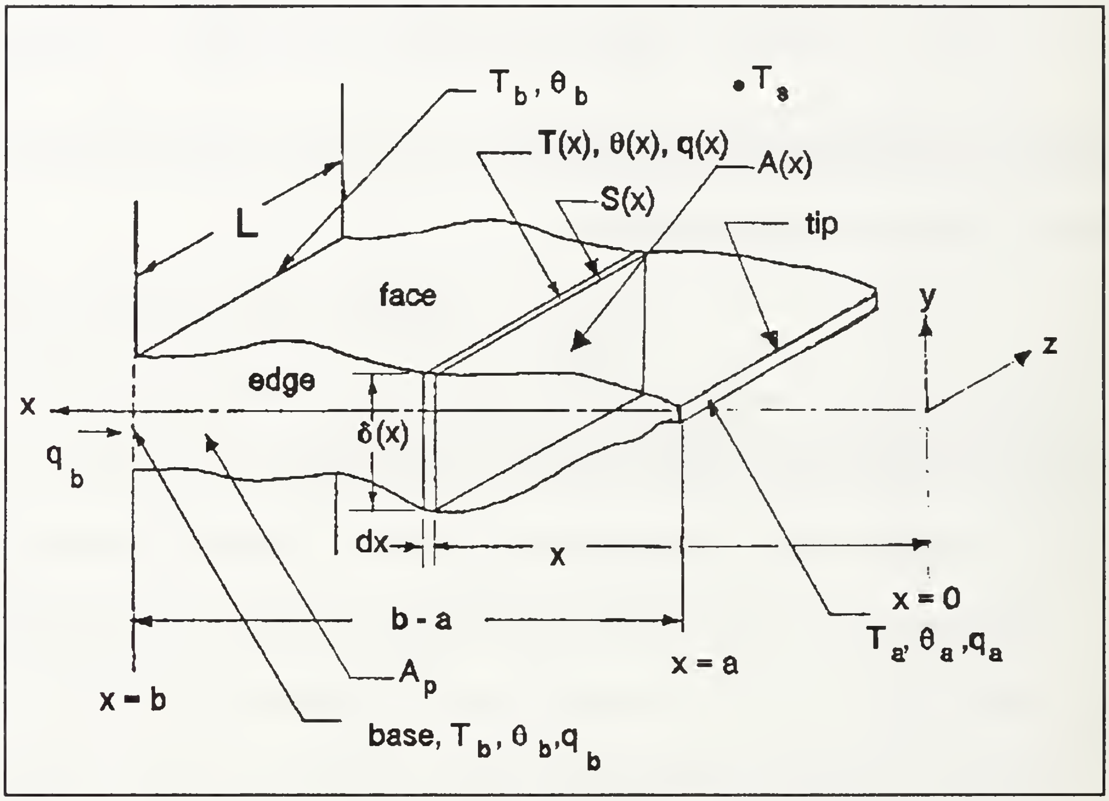

Figure 2.2 Arbitrary Symmetrical Longitudinal Fin[3]

Because the second Murray-Gardner assumption (Table 1) states that the material is homogeneous, both the emissivity and thermal conductivity are assumed to be constant and uniform

TABLE 1

THE MURRAY-GARDNER ASSUMPTIONS[4]

1. The heat flow into the fin and the temperature at any point on the fin remains constant with time.

2. The fin material is homogeneous, its thermal conductivity is the same in all directions and remains constant.

3. The heat transfer coefficient between the fin and the surrounding medium is uniform and constant over the entire surface of the fin.

4. The temperature of the medium surrounding the fin is uniform.

5. The fin width is so small compared with its height that temperature gradients across the fin width may be neglected.

6. The temperature at the base of the fin is uniform.

7. There are no heat sources within the fin itself.

8. Heat transfer to or from the fin is proportional to the temperature excess between the fin and the surrounding medium.

9. There is no contact resistance between fins in the configuration or between the fin at the base of the configuration and the prime surface.

10. The heat transferred through the outermost edge of the fin (the fin tip) is negligible compared to that through the lateral surfaces (faces) of the fin.

$$\epsilon(x,T) = \epsilon \tag{2.26}$$

$$k(T) = k \tag{2.27}$$

The contribution from convective dissipation from the faces of the fin is ignored because the arbitrary symmetrical longitudinal fin is operating in a space environment.

C. GENERALIZED DIFFERENTIAL EQUATION FOR A FIN

Combining the generalized differential equation (2.17) and equations (2.24) through (2.27), and assuming that there are no external heat inputs ($q_{i1} = q_{i2} = 0$), the steady state thermal energy balance for the differential element shown in Figure 2.2 is

$$\frac{d}{dx}\left(k\ L\ \delta(x)\frac{dT(x)}{dx}\right) - 2\ L\ \sigma\ \epsilon\ T(x)^4\ \frac{d\gamma}{dx} = 0 \tag{2.28}$$

The temperature excess $\theta(x)$ is defined as the temperature difference between the temperature at point x and the constant temperature of the surrounding environment T_s.

$$\theta(x) = T(x) - T_s \tag{2.29}$$

and from this, it is seen that

$$dT(x) = d\theta(x) \tag{2.30}$$

Applying the temperature excess equations (2.29) and (2.30), the equation for the steady state thermal energy balance for the differential element is rewritten as

$$\frac{d}{dx}\left[\delta(x)\,\frac{d\theta(x)}{dx}\right] - \frac{2\,h_r}{k}\,\theta(x)\,\frac{d\gamma}{dx} = 0 \qquad (2.31)$$

where h_r is defined as the radiative heat transfer coefficient. Because $T_s \approx 0$,

$$h_r = \sigma\,\epsilon\,\theta(x)^3 \qquad (2.32)$$

A fundamental concept of calculus is that the length of arc $d\gamma$ in a cartesian coordinate framework is

$$d\gamma = \left[\,dx^2 + dy^2\right]^{1/2} \qquad (2.33)$$

The derivative of the arc length is

$$\frac{d\gamma}{dx} = \lim_{\Delta x \to 0}\frac{\Delta\gamma}{\Delta x} = \pm\sqrt{1 + \left(\frac{dy}{dx}\right)^2} \qquad (2.34)$$

The + or - sign is to be taken according as the arc length γ increases or decreases as x increases.

Combining the equation for the steady state thermal energy balance differential element (2.31) and the arc length equation (2.34), the generalized differential equation for a longitudinal fin of arbitrary symmetrical profile is

$$\frac{d}{dx}\left[\delta(x)\,\frac{d\theta(x)}{dx}\right] - \frac{2\,h_r}{k}\,\theta(x)\,\left[1 + \acute{\delta}(x)^2\right]^{1/2} = 0 \qquad (2.35)$$

Equation (2.35) is a second order, linear, homogeneous differential equation with the boundary conditions

$$\frac{d\theta(\ x=a\ )}{dx} = 0 \tag{2.36}$$

and

$$\theta(\ x=b\ ) = \theta_b \tag{2.37}$$

D. LONGITUDINAL FIN OF RECTANGULAR PROFILE

The longitudinal fin of rectangular profile is illustrated in Figure 2.3. It should be noted that the origin of the x axis is located at the tip of the fin and not at the base of the fin. Because the width is constant over the entire profile, the width function δ(x) is simply

$$\delta(x) = \delta_b = \delta_a \tag{2.38}$$

The profile area A_p is written as

$$A_p = \int_{x=0}^{x=b} \delta(x)\ dx = \delta_b\ b \tag{2.39}$$

Inserting the width function (2.38) into the generalized differential equation for a fin equation (2.35) yields

$$\delta_b \frac{d^2\theta(x)}{dx^2} - \frac{2\ h_r}{k}\ \theta(x) = 0 \tag{2.40}$$

or

$$\frac{d^2\theta(x)}{dx^2} - m^2\,\theta(x) = 0 \qquad (2.41)$$

where

$$m = \sqrt{\frac{2\,h_r}{k\,\delta_b}} \qquad (2.42)$$

The radiative heat transfer coefficient h_r is assumed to be a constant for a very small subfin where $T_b \approx T_a$.

The general solution of the generalized differential equation (2.41) is

$$\theta(x) = C_1\,e^{m\,x} + C_2\,e^{-m\,x} \qquad (2.43)$$

where C_1 and C_2 are arbitrary integration constants that are evaluated by applying the initial condition of equation (2.22).

$$\theta(b) = \theta_b = C_1\,e^{m\,b} + C_2\,e^{-m\,b} \qquad (2.44)$$

The heat flow, q_b, into the base of the fin is found by inserting the general solution of the generalized differential equation (2.43) and the cross sectional area perpendicular to the heat flow by conduction (2.25) into Fourier's Law of

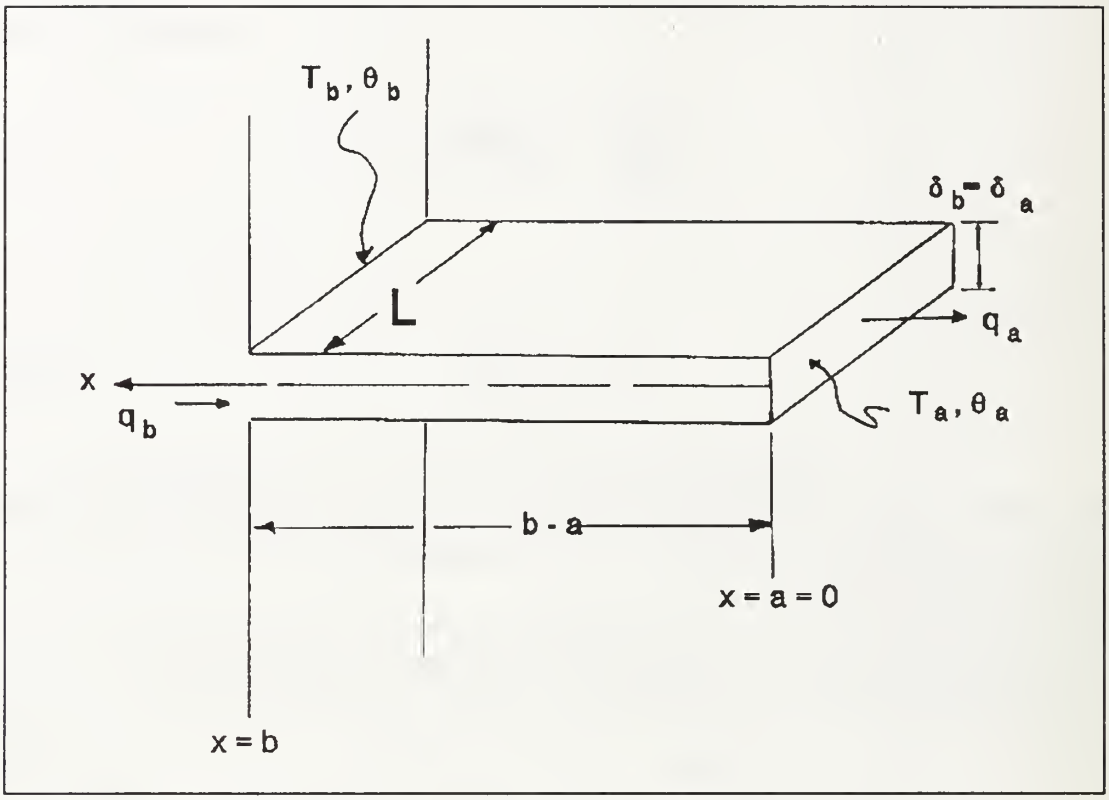

Figure 2.3 Longitudinal Fin of Rectangular Profile

Conduction, equation (2.4).

$$q(x) = L\ \delta_b\ k \left(\frac{d}{dx}\left[C_1\ e^{m\,x} + C_2\ e^{-m\,x}\right]\right) \tag{2.45}$$

Simplifying and evaluating at x = b with the initial condition of equation (2.23), equation (2.45) becomes

$$q_b = L\ \delta_b\ k\ m \left(C_1\ e^{m\,b} - C_2\ e^{-m\,b} \right) \tag{2.46}$$

Solving for C_1 and C_2 yields

$$C_1 = \frac{e^{-m\,b}}{2}\,\theta_b + \frac{e^{-m\,b}}{2\,Y_b}\,q_b \tag{2.47}$$

and

$$C_2 = \frac{e^{m\,b}}{2}\,\theta_b - \frac{e^{m\,b}}{2\,Y_b}\,q_b \tag{2.48}$$

where

$$Y_b = L\,(\,2\,h_r\,k\,\delta_b)^{1/2} \tag{2.49}$$

is defined as the characteristic thermal admittance Y_b of the fin. Likewise the characteristic thermal impedance, Z_b, of the fin is the reciprocal of Y_b

$$Z_b = \frac{1}{Y_b} = \frac{1}{L}\,(\,2\,h_r\,k\,\delta_b\,)^{-1/2} \tag{2.50}$$

An examination of equations (2.43) through (2.48) reveals that only two of the four variables are independent and that the specification of any two variables allows for the determination of the remaining two. Combining equations (2.43) through (2.48) yields the two port inverse thermal transmission matrix for a longitudinal fin of rectangular profile

$$\begin{bmatrix}\theta_a\\ q_a\end{bmatrix} = \begin{bmatrix}\cosh mb & -Z_b \sinh mb\\ -Y_b \sinh mb & \cosh mb\end{bmatrix}\begin{bmatrix}\theta_b\\ q_b\end{bmatrix} \tag{2.51}$$

where the inverse thermal transmission matrix is defined as

$$T^{-1} = \begin{bmatrix} \cosh mb & -Z_b \sinh mb \\ -Y_b \sinh mb & \cosh mb \end{bmatrix} \tag{2.52}$$

The thermal transmission matrix is defined as the inverse of the inverse thermal transmission matrix.

$$T = T^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{2.53}$$

where A, B, C, and D are the elements of the thermal transmission matrix. The thermal transmission matrix representation is

$$\begin{bmatrix} \theta_b \\ q_b \end{bmatrix} = \begin{bmatrix} \cosh mb & Z_b \sinh mb \\ Y_b \sinh mb & \cosh mb \end{bmatrix} \begin{bmatrix} \theta_a \\ q_a \end{bmatrix} \tag{2.54}$$

E. LONGITUDINAL FIN OF TRAPEZOIDAL PROFILE

The longitudinal fin of trapezoidal profile is shown in Figure 2.4. Note that the origin of the axis is not at the tip of fin but located to the right of the tip where x = 0.

The width function δ(x) for this profile is

$$\delta(x) = 2\,x \tan \phi \tag{2.55}$$

where ϕ is defined as

$$\phi = \arctan\left[\frac{\delta_b}{2\,(b - a)}\right] \tag{2.56}$$

The profile area A_p is

$$A_p = \int_{x=a}^{x=b} \delta(x)\, dx = \frac{(b - a)}{2}\,(\delta_b + \delta_a) \tag{2.57}$$

Inserting the width function (2.55) into the generalized differential equation for a fin [Equation (2.35)] yields

$$2\,x\,\tan(\phi)\,\frac{d^2\theta(x)}{dx^2} + 2\,\tan(\phi)\,\frac{d\theta(x)}{dx} - \frac{2\,h_r}{k\,\cos(\phi)}\,\theta(x) = 0 \tag{2.58}$$

or

$$x\,\frac{d^2\theta(x)}{dx^2} + \frac{d\theta(x)}{dx} - m^2\,\theta(x) = 0 \tag{2.59}$$

where

$$m = \left[\frac{h_r}{k\,\sin(\phi)}\right]^{1/2} \tag{2.60}$$

Equation (2.59) represents a second order, linear, homogenous differential equation. The general solution is expressed in terms of modified Bessel functions. The general

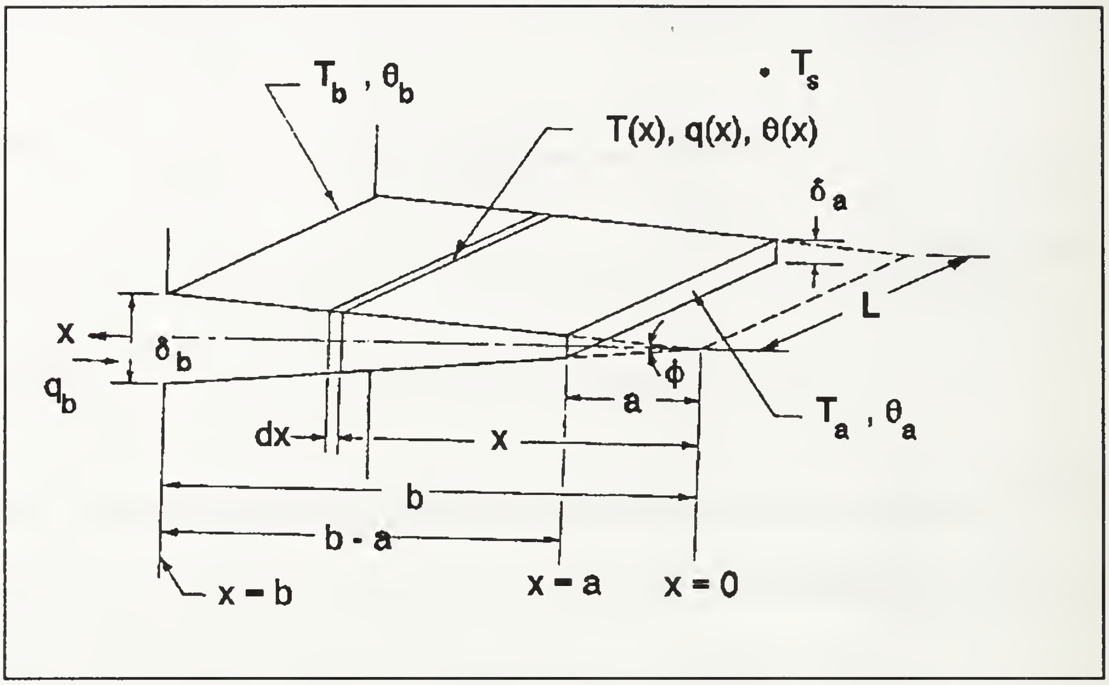

Figure 2.4 Longitudinal Fin of Trapezoidal Profile[5]

solution to equation (2.59) is

$$\theta(x) = C_1\, I_o(\,2\,m\,x^{1/2}\,) + C_2\, K_o(\,2\,m\,x^{1/2}\,) \tag{2.61}$$

The following transformation is made

$$u = 2\,m\,x^{1/2} \tag{2.62}$$

such that at x = b

$$u_b = 2\,m\,b^{1/2} \tag{2.63}$$

and at x = a

$$u_a = 2\ m\ a^{1/2} \tag{2.64}$$

After applying the boundary conditions of equations (2.36) and (2.37) and making the transformations of equations (2.63) and (2.64), the general solutions become

$$\theta_b = C_1\ I_0(u_b) + C_2\ K_0(u_b) \tag{2.65}$$

and

$$q_b = \frac{2\ k\ \delta_b\ L\ m^2}{u_b}\left[C_1\ I_1(u_b) - C_2\ K_1(u_b)\right] \tag{2.66}$$

Simultaneous solution of the general solution equations (2.65) and (2.66) yield

$$C_1 = u_b\left[K_1(u_b)\ \theta_b + \left(\frac{u_b}{2\ k\ \delta_b\ L\ m^2}\right) K_o(u_b)\ q_b\right] \tag{2.67}$$

and

$$C_2 = u_b\left[I_1(u_b)\ \theta_b - \left(\frac{u_b}{2\ k\ \delta_b\ L\ m^2}\right) I_o(u_b)\ q_b\right] \tag{2.68}$$

Substituting and rearranging the general solution equations (2.65) and (2.66) and the equations for C_1 and C_2 (2.67) and (2.68), the elements of the thermal transmission matrix for the longitudinal fin of trapezoidal profile are written as:

$$A = u_a \, [\, I_1(u_a) \, K_o(u_b) \, + \, I_o(u_b) \, K_1(u_a)] \tag{2.69}$$

$$B = \frac{u_b^{\,2}}{2 \; k \; \delta_b \; L \; m^2} \; \left[I_0(u_b) K_0(u_a) \, - \, I_0(u_a) \, K_0(u_b) \right] \tag{2.70}$$

$$C = 2 \; k \; \delta_a \; L \; m^2 \left(\frac{u_b}{u_a} \right) [\, I_1(u_b) \, K_1(u_a) \, - \, I_1(U_a) \, K_1(u_b)] \tag{2.71}$$

$$D = u_b \, [\, I_0(u_a) \, K_1(u_b) \, + \, I_1(u_b) \, K_0(u_a)] \tag{2.72}$$

F. LONGITUDINAL FIN OF TRIANGULAR PROFILE

The longitudinal fin of triangular profile shown in Figure 2.5 is a special case of the longitudinal fin of trapezoidal profile. There are many similarities between the longitudinal fin of triangular and trapezoidal profiles. The differential equation for the longitudinal fin of triangular profile is the same as the differential equation for the longitudinal fin of trapezoidal profile equation (2.59).

$$x \, \frac{d^2\theta(x)}{dx^2} \, + \, \frac{d\theta(x)}{dx} \, - \, m^2 \, \theta(x) \, = \, 0 \tag{2.73}$$

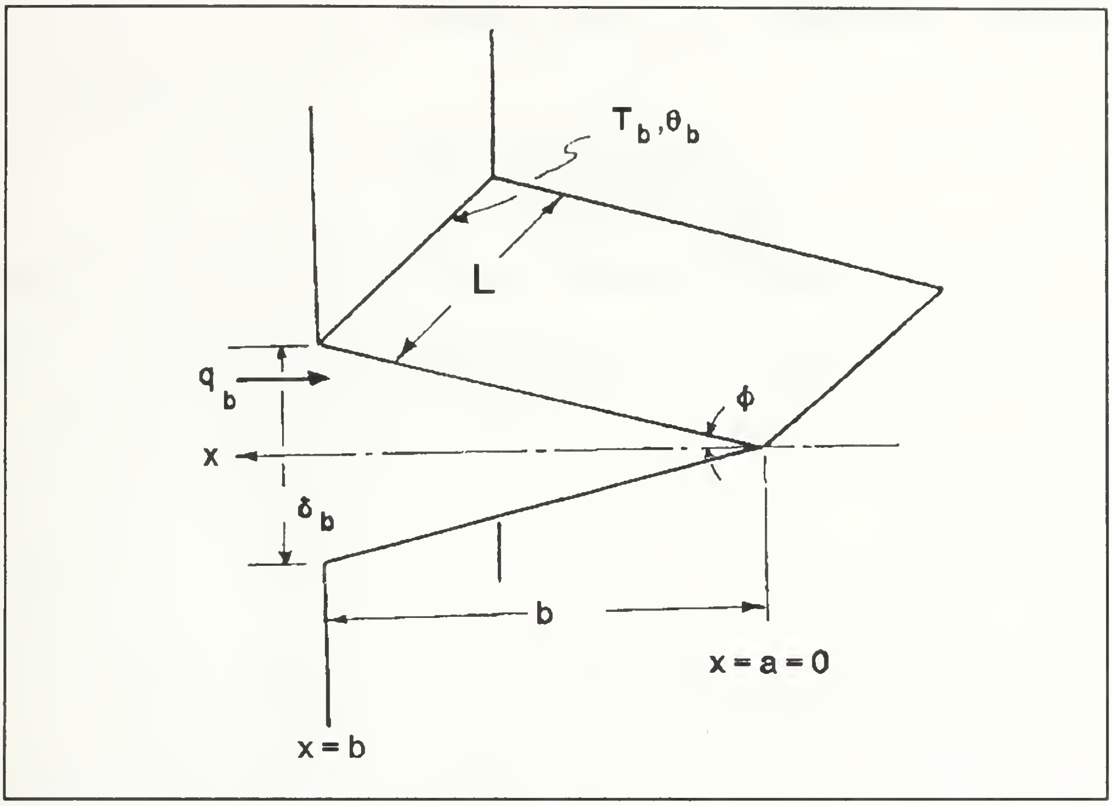

Figure 2.5 Longitudinal Fin of Triangular Profile[6]

where

$$m = \left[\frac{h_r}{k \sin(\phi)}\right]^{1/2} \tag{2.74}$$

and

$$\phi = \arctan\left[\frac{\delta_b}{2\ b}\right] \tag{2.75}$$

The profile area A_p is

$$A_p = \int_{x=0}^{x=b} \delta(x)\, dx = \frac{b}{2}\,\delta_b \tag{2.76}$$

However the evaluation of the arbitrary constants from the boundary conditions differ between the triangular and the trapezoidal profiles. As the value of x approaches zero the temperature excess at the tip θ_a must be equal to some finite value, therefore

$$I_o(\, u{=}0\,) = 1 \tag{2.77}$$

In order to physically obtain a finite temperature at $x_a = u_a = 0$, C_2 in equation (2.61) must be equal to zero because $K_2(0)$ is unbounded. The unbounded condition allows cancellation of the $K_o(u)$ terms from the general solutions for temperature excess and heat flow.

After cancellation and rearranging the general solutions for the longitudinal fin of trapezoidal profile equations (2.65) and (2.66), the general solution for the longitudinal fin of triangular profile becomes

$$q_b = \frac{2\,k\,\delta_b\,L\,m^2}{u_b}\,\frac{I_1(u_b)}{I_0(u_b)}\,\theta_b \tag{2.78}$$

As a result of the zero tip width, the edge parameters do not depend on the base parameters. The thermal input admittance element for the longitudinal fin of triangular profile is

$$Y_i = \frac{q_b}{\theta_b} = \frac{2\ k\ \delta_b\ L\ m^2}{u_b} \frac{I_1(u_b)}{I_0(u_b)} \tag{2.79}$$

III. CASCADE ANALYSIS ALGORITHM

A. GENERALIZED TWO PORT TRANSMISSION LINE MODEL

The radiating fin is a distributed heat flow configuration which is analogous to the transmission line which is a distributed electrical network. The general two port transmission line model is shown in Figure 3.1. The model is described by two input terminals. At point "a" there is an applied voltage V_a and a current I_a flows into the

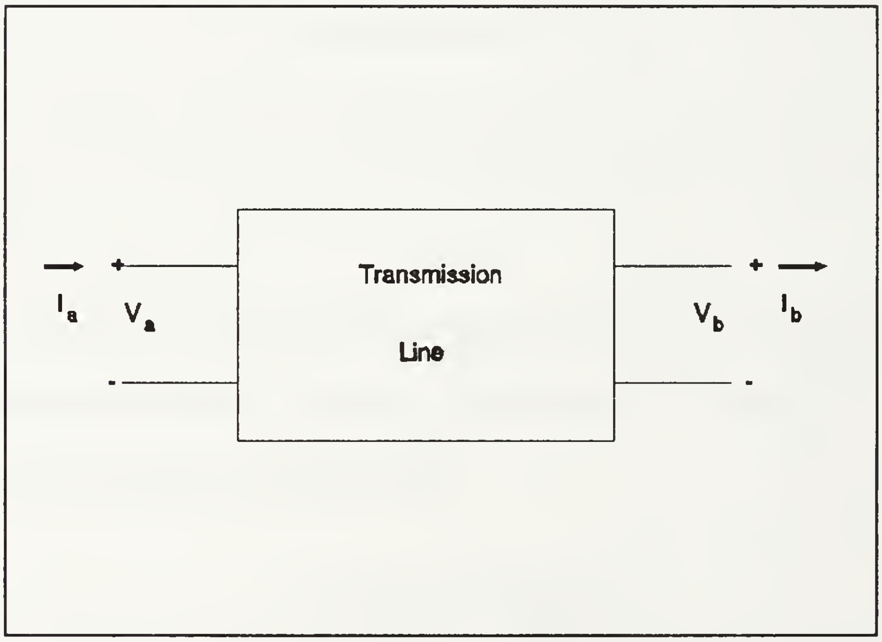

Figure 3.1 Two Port Transmission Line Model

model. There is also a pair of terminals at point "b" where a voltage V_b occurs and a current I_b flows out of the terminals. The two port model of the transmission line can consist of any arrangement of passive elements within the two port but with no active current or voltage sources within the two port.

It can be shown from network theory that the two port transmission line model in Figure 3.1 can be represented by the following equations:

$$V_a = A\ V_b + B\ I_b \tag{3.1}$$

$$I_a = C\ V_b + D\ I_b \tag{3.2}$$

where the determinant of the ABCD matrix defined by

$$\begin{bmatrix} V_a \\ I_a \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} V_b \\ I_b \end{bmatrix} \tag{3.3}$$

is equal to unity.

$$A\ D - B\ C = 1 \tag{3.4}$$

The values A, B, C, and D are constants and are defined as the elements of the transmission matrix.

The electrical transmission line analogy can be directly applied to an equivalent thermal transmission line model where the heat flow is analogous to current and the temperature is analogous to voltage. Equations (3.1) and (3.2) are rewritten in the equivalent thermal transmission line model as:

$$T_a = A\ T_b + B\ q_b \tag{3.5}$$

$$q_a = C\, T_b + D\, q_b \tag{3.6}$$

B. CASCADE ALGORITHM

The thermal transmission parameters are very useful in describing two port networks connected in a cascaded arrangement. It is an exercise in elementary two port theory. Examine the two equations for the two networks shown in Figure 3.2.

$$\begin{bmatrix} \theta_{b_1} \\ q_{b_1} \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} \theta_{a_1} \\ q_{a_1} \end{bmatrix} \tag{3.7}$$

and

$$\begin{bmatrix} \theta_{b_2} \\ q_{b_2} \end{bmatrix} = \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \begin{bmatrix} \theta_{a_2} \\ q_{a_2} \end{bmatrix} \tag{3.8}$$

Continuity for the configuration in Figure 3.2 dictates that

$$\theta_{b_2} = \theta_{a_1} \tag{3.9}$$

and

$$q_{b_2} = q_{a_1} \tag{3.10}$$

which can be represented as

$$\begin{bmatrix} \theta_{b_2} \\ q_{b_2} \end{bmatrix} = \begin{bmatrix} \theta_{a_1} \\ q_{a_1} \end{bmatrix} \tag{3.11}$$

The overall thermal transmission parameter matrix for the two networks cascaded in Figure 3.2 is by definition

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \tag{3.12}$$

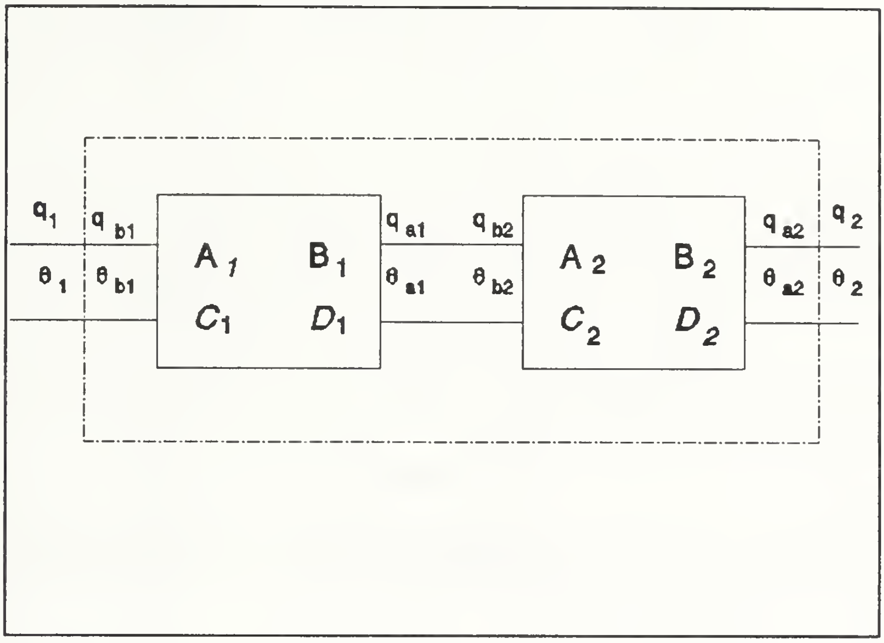

Figure 3.2 Cascaded Network

and the thermal transmission matrix for a cascaded network of n networks is

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \cdots \begin{bmatrix} A_n & B_n \\ C_n & D_n \end{bmatrix} \tag{3.13}$$

Equation (3.13) is an expression of what is referred to as the cascade algorithm.

C. COMPUTER ANALYSIS

The cascade algorithm expression (3.13) can be applied to a fin divided into n subfins each having the same length. Converting the temperature excess notation back to the temperature notation ($T_s \approx 0$), the cascade algorithm expression (3.13) is written as

$$\begin{bmatrix} T_b \\ q_b \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \cdots \begin{bmatrix} A_n & B_n \\ C_n & D_n \end{bmatrix} \begin{bmatrix} T_a \\ q_a \end{bmatrix} \tag{3.14}$$

The base temperature T_b and the base heat flow q_b are the conditions at the base and the tip temperature T_a and tip heat flow q_a are the conditions at the edge.

A problem arises because the tip temperature T_a and the base heat flow q_b are not known or not specified. The cascade algorithm expression (3.14) can not be solved directly because it is essentially a linear transformation that maps tip conditions to base conditions. However, a solution can be calculated indirectly. If one makes an initial estimation for the tip temperature T_a, it can be used to calculate the radiant heat flow q_a out of the tip of the fin.

$$q_a = \sigma \, \epsilon \, \delta_a \, L \, T_a^4 \tag{3.15}$$

Thus T_a and q_a are the starting values for the computations using the cascade algorithm expression (3.14). At the conclusion of the process, a comparison involving all n subfins is then made between the known or specified base temperature T_b and the calculated base temperature. If the comparison indicates an error, the tip temperature T_a is changed and the iterative process is repeated and continues until the error between the presumed known base temperature T_b and the calculated base temperature are within an acceptable range. Figure 3.4 provides a complete flow chart of the cascade algorithm.

Further amplification concerning the radiative heat transfer coefficient needs to be made. Recalling equation (2.32) where

$$h_r = \sigma \epsilon T(x)^3 \tag{3.16}$$

it is not clear as to what temperature, T(x), should be applied to the computation of h_r and hence to the cascade algorithm expression (3.14). The left and right faces of the differential element in Figure 3.3 have an average temperature.

$$T_{av} = \frac{T_a + T_b}{2} \tag{3.17}$$

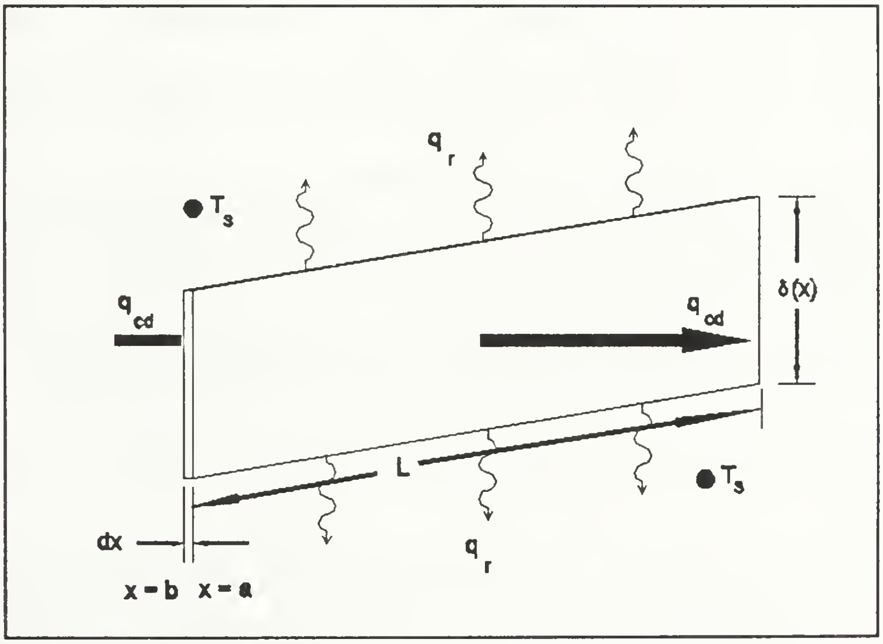

Figure 3.3 Differential Element

and as long as T_a and T_b are not vastly different, equation (3.16) can be written using T_{av}

$$h_r = \sigma \epsilon T_{av}^3 \tag{3.18}$$

The environmental parameter V is introduced for the case in which there are portions of the surrounding environment at some temperature other than $T_s \approx 0$. An example of this is in the interception of thermal radiation from a nearby solar panel at some temperature not near absolute zero. The environmental parameter is defined as

$$V = \frac{K_2}{2 \sigma \epsilon T_b^4} \tag{3.19}$$

where the parameter K_2 allows for the application of external heat inputs from solar, E_S, or terrestrial, E_T, sources or both.

$$K_2 = E \alpha \tag{3.20}$$

The environmental parameter is equal to zero for the free space case and to some number greater than zero but less than one for the non free space case. Substituting equation (3.19) into (3.18), the radiant heat transfer coefficient becomes

$$h_r = \sigma \epsilon \left[T_{av}^3 - \frac{K_2}{2 T_{av}} \right] \tag{3.21}$$

External heat inputs (q_{i1} and q_{i2}) from solar input E_S and terrestrial input E_T are now applied to the cascade algorithm expression (3.14) through the application of the environmental parameter V.

D. PROGRAM DESCRIPTION

The computer code consists of three analysis programs for each of the profiles considered, and a MAIN Program that is used for general housekeeping. The PERFORMANCE Analysis Program evaluates an entered set of parameters returning the characteristics (volume, profile area A_p, profile number PN, base heat flow q_b, edge temperature T_a, and the fin efficiency η) of the fin along with the thermal transmission parameter matrix. The PROFILE AREA Analysis Program also evaluates an entered set of parameters but it returns a data set of 100 fins having a constant profile area A_p with the base fin width δ and length b which are used as the input variables. The OPTIMIZE Analysis Program is similar to the PERFORMANCE Analysis Program with the exception that it returns the optimum fin geometry which results in the minimum mass and maximum base heat flow fin.

The MAIN Program provides menu driven access to the various analysis programs. The user has the ability to set the computational main program error coefficient. The user also has the ability to set the save data switch which controls data flow into an ASCII formatted disk file. Commas are used for the data delimiter which allows file importation into MATLAB, Word Perfect, Draw Perfect, and Math CAD for additional analysis or for plotting. An additional feature provided is the unit conversion module. Table 2 list the available units that are provided to the user by the MAIN Program. Figure 3.4 provides a flow chart of the cascade algorithm expression (3.14) which is common to all of the analysis programs.

The PERFORMANCE Analysis Program requires the following input conditions to be entered:

- base temperature
- density
- emissivity
- environmental parameter
- fin base width
- fin edge width[1]
- fin depth
- fin length
- thermal conductivity

The program calculates and returns the following values:

- base heat flow
- edge temperature
- fin efficiency
- profile area
- profile number
- thermal transmission parameter matrix
- total mass

The fin efficiency η and the profile number PN are extensively referred to in literature and are provided for use in making comparisons. The fin efficiency and the profile number are defined as

$$\eta = \frac{q_b}{2 \sigma \epsilon b L T_b^4} \qquad (3.22)$$

[1]Applies for only the longitudinal fin trapezoidal profile.

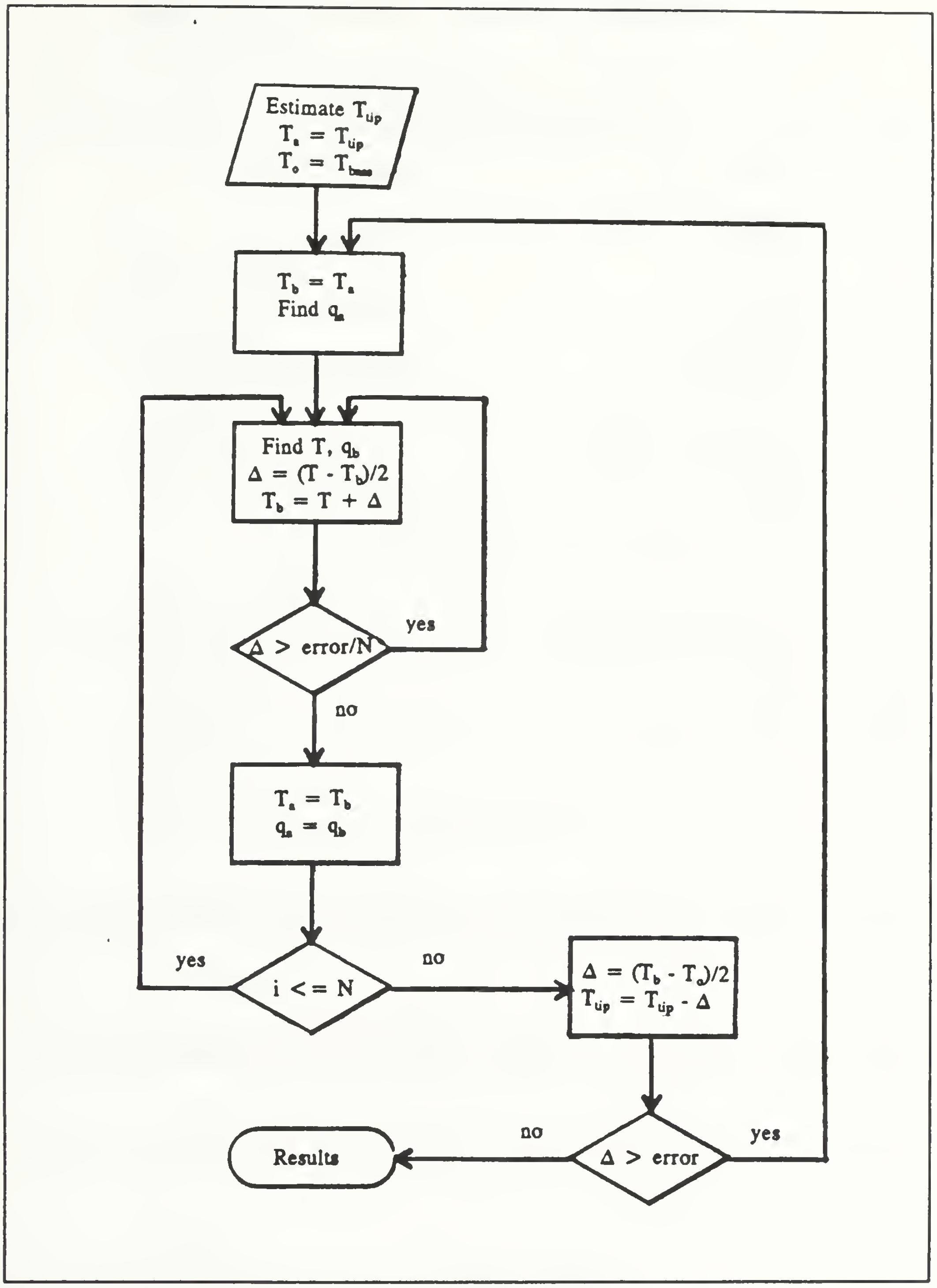

Figure 3.4 Cascade Algorithm Flow Chart

TABLE 2

UNITS AVAILABLE IN THE MAIN PROGRAM

Admittance	• watts / °C, BTU / hr °F
Area	• m^2, cm^2, ft^2, in^2, mm^2
Heat	• watts, kw, BTU / sec, BTU / hr
Impedance	• °C / watts, °F hr / BTU
Length and Depth	• m, cm, ft, in, mm
Mass	• kg, grams, lbm, oz
Temperature	• K, °C, °F, °R
Thermal Conductivity	• watts / m K, BTU / ft hr °F
Width	• m, cm, ft, in, mm

$$PN = \frac{2\ \sigma\ \epsilon\ b^2\ T_b^{\ 3}}{k\ \delta_b} \quad \text{(trapezoidal)} \quad (3.23)$$

$$PN = \sqrt{\frac{\sigma\ \epsilon\ b^2\ T_b^{\ 3}}{k\ \delta_b}} \quad \text{(rectangular)} \quad (3.24)$$

If the save switch is switched on, the following differential element conditions are written to a disk file:

- base heat flow
- base temperature
- distance from the tip of the fin

The OPTIMIZE Analysis Program requires the following input conditions to be entered:

- base temperature
- density
- emissivity
- environmental parameter
- fin edge width[2]
- fin width
- thermal conductivity

The program calculates and returns the following optimum values:

- base heat flow
- edge temperature
- fin base width
- fin efficiency
- fin length
- profile area
- profile number
- thermal transmission parameter matrix
- total mass

The program saves the same quantities that the PERFORMANCE Analysis Program saves depending on the condition of the save switch.

The PROFILE AREA Analysis Program requires the following input conditions

[2]Applies for only the longitudinal fin trapezoidal profile.

to be entered:

- base temperature
- density
- emissivity
- environmental parameter
- profile area
- starting fin base width value
- ending fin base width value
- taper ratio[3]
- fin depth
- thermal conductivity

The program calculates and writes to a disk file the following parameters for 100 fins having the same profile area A_p within the specified width range:

- base heat flow
- base width
- base temperature
- edge temperature
- fin efficiency
- thermal transmission parameter matrix
- total mass

The analysis programs all use the thermal transmission matrices computed in Chapter 2 in the cascade algorithm with the exception of the longitudinal fin of triangular profile. The triangular and trapezoidal profile analysis are very similar. The analysis program uses the thermal admittance equation (2.79) to compute the base conditions for the first subfin having a thickness of δ_a. The subfin is located at the tip of the fin. The base conditions are then applied and used as the tip conditions in the thermal transmission matrix for the trapezoidal profile (2.69) through (2.72). The

[3]Applies for only longitudinal fin of trapezoidal profile.

cascade algorithm is then used to calculate the thermal transmission matrices for the remaining 99 subfins in order to finally determine the overall thermal transmission matrix and the base conditions. The trapezoid that is being used in the calculation has an almost zero tip width compared to the base width such that

$$\delta_b = 100\,\delta_a \tag{3.25}$$

Borland Turbo C++ was used to write the computer code. The computer code listing appears in Appendix A. Full compiler optimization was selected in order to decrease the run times. Average run times on an IBM 486/487 machine appear in Table 3.

TABLE 3

AVERAGE PROGRAM RUN TIMES

	PERFORMANCE (sec)	PROFILE AREA (sec)	OPTIMIZE (sec)
Rectangular	4	97	4
Trapezoidal	4	600	4
Triangular	5	625	5

IV. PROGRAM ANALYSIS RESULTS AND DISCUSSION

A. PROGRAM ERROR TOLERANCE

In order to allow for meaningful interpretation, an understanding of the accuracy of the results produced from the cascade algorithm is vital. A test to show convergence is used to illustrate the approximate relative error.

Table 4 provides an illustration of how the error coefficient in the MAIN Program affects the accuracy of the program results. Table 4 is produced by

TABLE 4

PROGRAM ERROR TOLERANCE

Program Error Coeff %	Ta (K)	≈ % error	q_b (watts)	≈ % error
1.000	282.18	0.0106	926.28	2.0918
0.500	282.24	0.0106	931.83	1.5052
0.100	282.19	0.0071	946.36	0.0307
0.050	282.22	0.0035	946.94	0.0920
0.010	282.22	0.0035	946.10	0.0032
0.005	282.21	0.0000	946.05	0.0021
0.001	282.21	0.0000	946.07	0.0000

running the same input parameters many times while decreasing the error coefficient in the MAIN Program. The PERFORMANCE Analysis Program for the longitudinal fin of rectangular profile was used. The Table 4 results are applicable for any of the analysis programs because of the commonality of the cascade algorithm. As the error coefficient in the MAIN Program decreases, the output value converges. The approximate relative error ε_r is defined as

$$\varepsilon_r \approx \left[\frac{True\ Value\ -\ Approximate\ Value}{True\ Value}\right] 100 \tag{4.1}$$

The exact true value is unknown, however the value produced when the error coefficient in the MAIN Program is equal to 0.001 can be assumed to be approximately equal to the exact true value.

Examining the results illustrated in Table 4, one can conclude that when the error coefficient in the MAIN Program is less than or equal to 0.1 the approximate relative error observed in the output is generally much less than the error coefficient in the main program.

B. GRAPHICAL EXAMPLES

The purpose of this section is to describe and provide examples on how each of the three analysis programs can be used. The computer code listings are provided in Appendix A.

1. Example One: Longitudinal Fin of Rectangular Profile Analysis

A longitudinal fin of rectangular profile is 0.5 meter long, 1.0 centimeter thick, and 1.0 meter deep. The fin is made of aluminum (k = 209.4 watt / m K,

$\rho = 2.6$ kg / dm^3, and $\epsilon = 0.85$). If the base temperature is 400 K, what are the tip, midpoint, and base conditions (error $< 0.1\%$) both in the free space and non free space environments?

Because the geometry of the fin is already known, the PERFORMANCE Analysis Program is used for the evaluation. The save switch in the MAIN Program must be placed in the save position in order to later retrieve the midpoint conditions from the disk file. The computer output display for the free space environment (V=0) appears in Table 5.

Figures 4.1 and 4.2 provide a detailed graphical representation of the required conditions both in the free space and non free space environments. The base conditions are read from where the length is equal to 50 centimeters and the tip conditions are read from where the length is equal to 0 centimeter. In the free space environment the tip, midpoint, and base conditions for this particular fin geometry are listed in Table 6.

TABLE 5

PERFORMANCE ANALYSIS OUTPUT DISPLAY

```
   THE LONGITUDINAL FIN OF RECTANGULAR PROFILE

                 Density = 2.60 kg / dm^3
               Fin Depth = 1.00 m
              Total Mass = 13.00 kg
              Fin Length = 0.50 m
               Fin Width = 1.00 cm
              Emissivity = 0.85
            Profile Area = 50.00 cm^2
          Profile Number = 0.607
          Base Heat Flow = 732.85 watts
          Fin Efficiency = 0.594
        Base Temperature = 400.07 K
        Edge Temperature = 324.45 K
    Thermal Conductivity = 209.40 watts / m K
 Environmental Parameter = 0.00

          Transmission Parameter Matrix

A = 1.2287                B = 0.2583 deg C / watt
B = 2.2374 watt / deg C   D = 1.2842
```

TABLE 6

FIN CONDITIONS IN FREE SPACE

Location	Temp (K)	q (watts)
Base	400.07	732.85
Midpoint	341.66	292.33
Tip	324.45	10.68

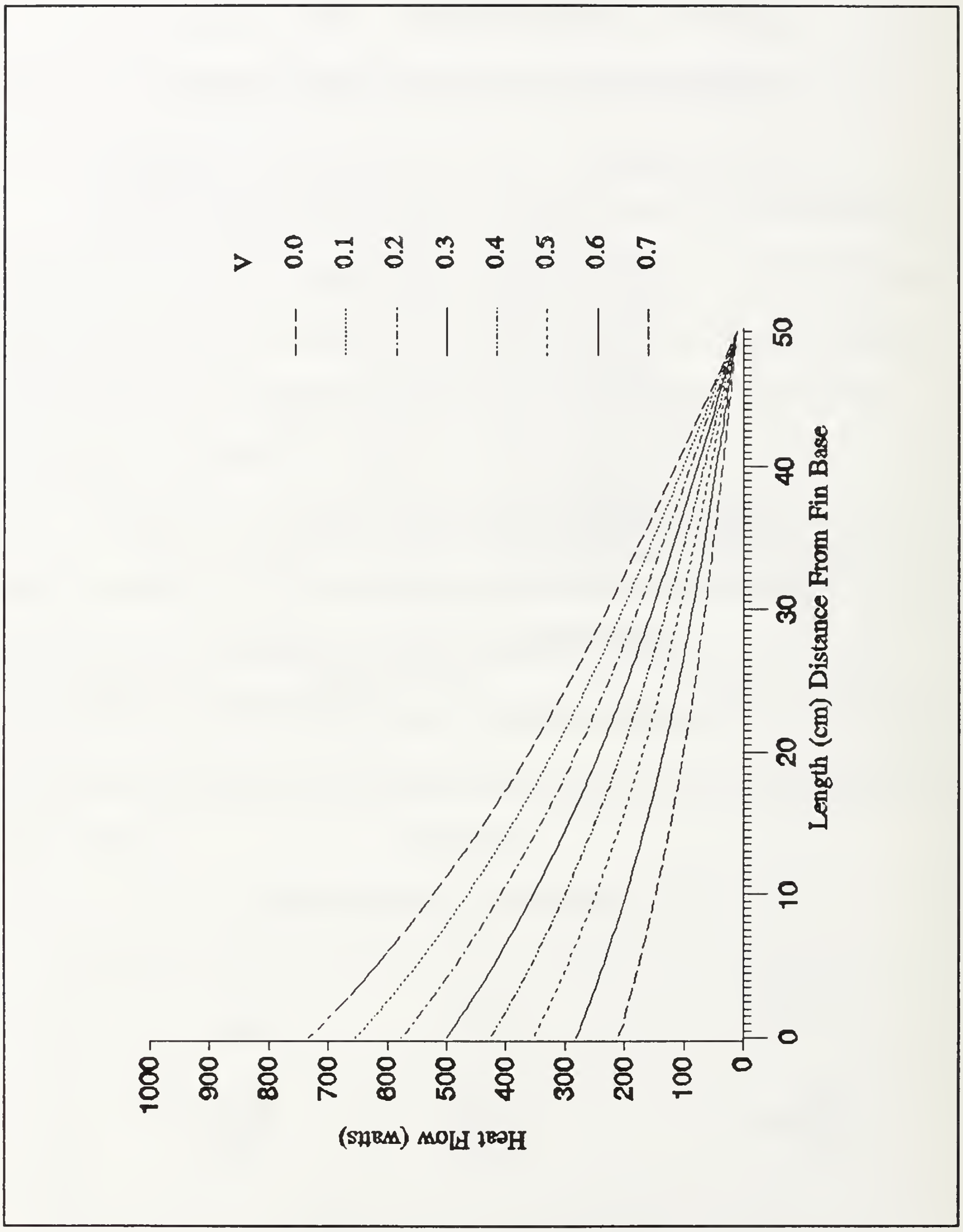

Figure 4.1 Heat Dissipation From A Rectangular Profile Fin

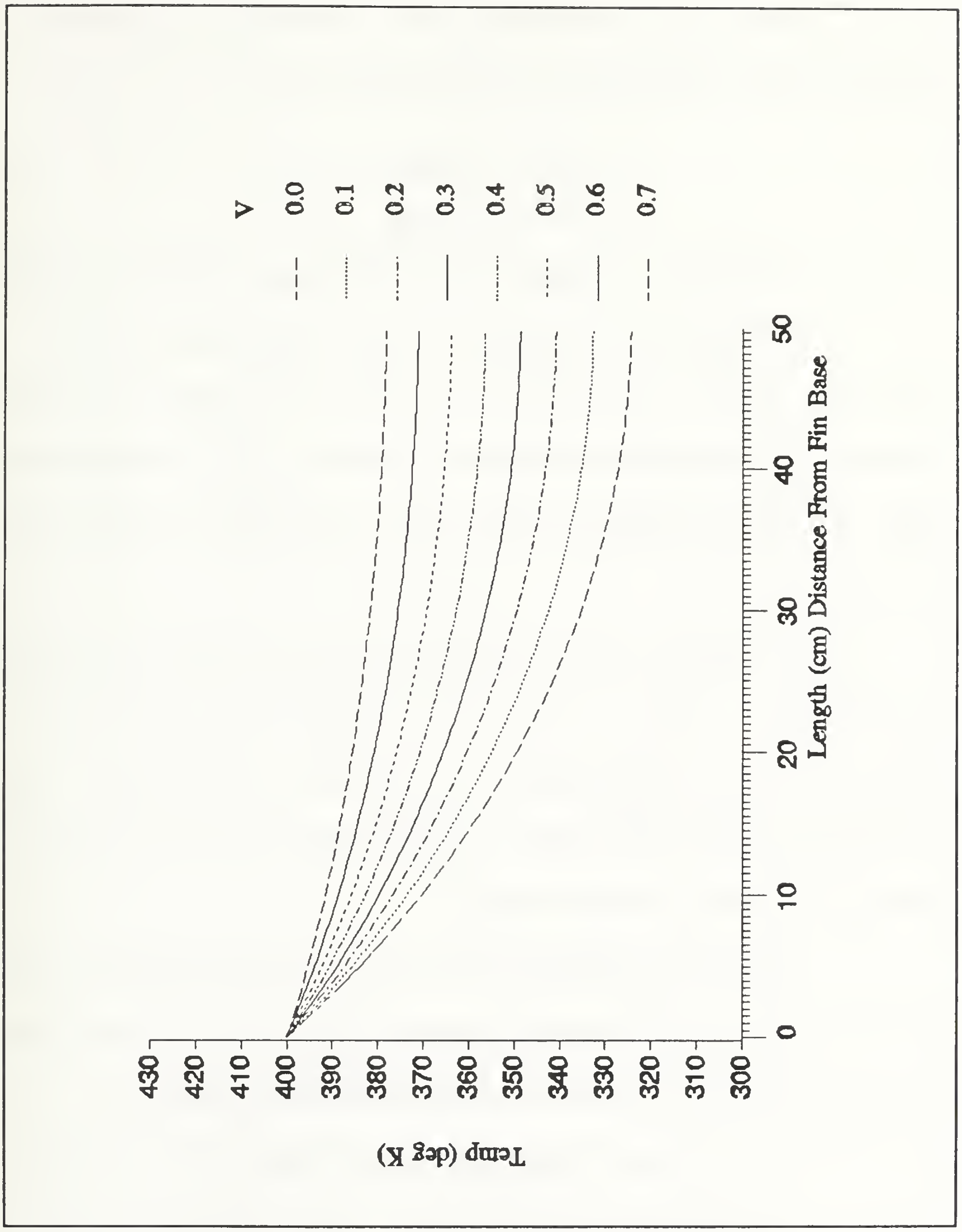

Figure 4.2 Temperature Distribution For A Rectangular Profile Fin

Equation (3.4) is used below to verify that the determinant of thermal transmission matrix listed in Table 5 is indeed equal to unity.

$$\det \begin{vmatrix} 1.2287 & 0.2583 \\ 2.2374 & 1.2842 \end{vmatrix} \approx 1.0000 \tag{4.2}$$

2. Example Two: Longitudinal Fin of Rectangular Profile Area Analysis

A longitudinal fin of rectangular profile 1.0 meter deep is made of aluminum ($k = 209.4$ watt / m K, $\rho = 2.6$ kg / dm^3, and $\epsilon = 0.85$). It is required to remove a minimum of 500 watts of heat from the base of a fin at a temperature of 400 K. Determine the geometry of the fin using several values for the profile area A_p.

The PROFILE AREA Analysis Program is used because the geometry of the fin is unknown. Figure 4.3 provides a graphical representation which shows the constant profile area A_p relationship as a function of width versus heat flow. Any of the dimensions listed in Table 7 would meet the minimum requirement to remove the 500 watts of heat from the base of the fin.

Examining Figure 4.3, one can see that the maximum heat flow occurs at a particular width value. More heat is dissipated as the profile area A_p is increased. The value of the width at the point of maximum heat flow also increases as the profile area A_p increases. The point of maximum heat flow is considered as the optimum geometry (maximum heat flow, minimum mass) for the particular profile area A_p. A summary of the optimum geometry points extrapolated from the data files

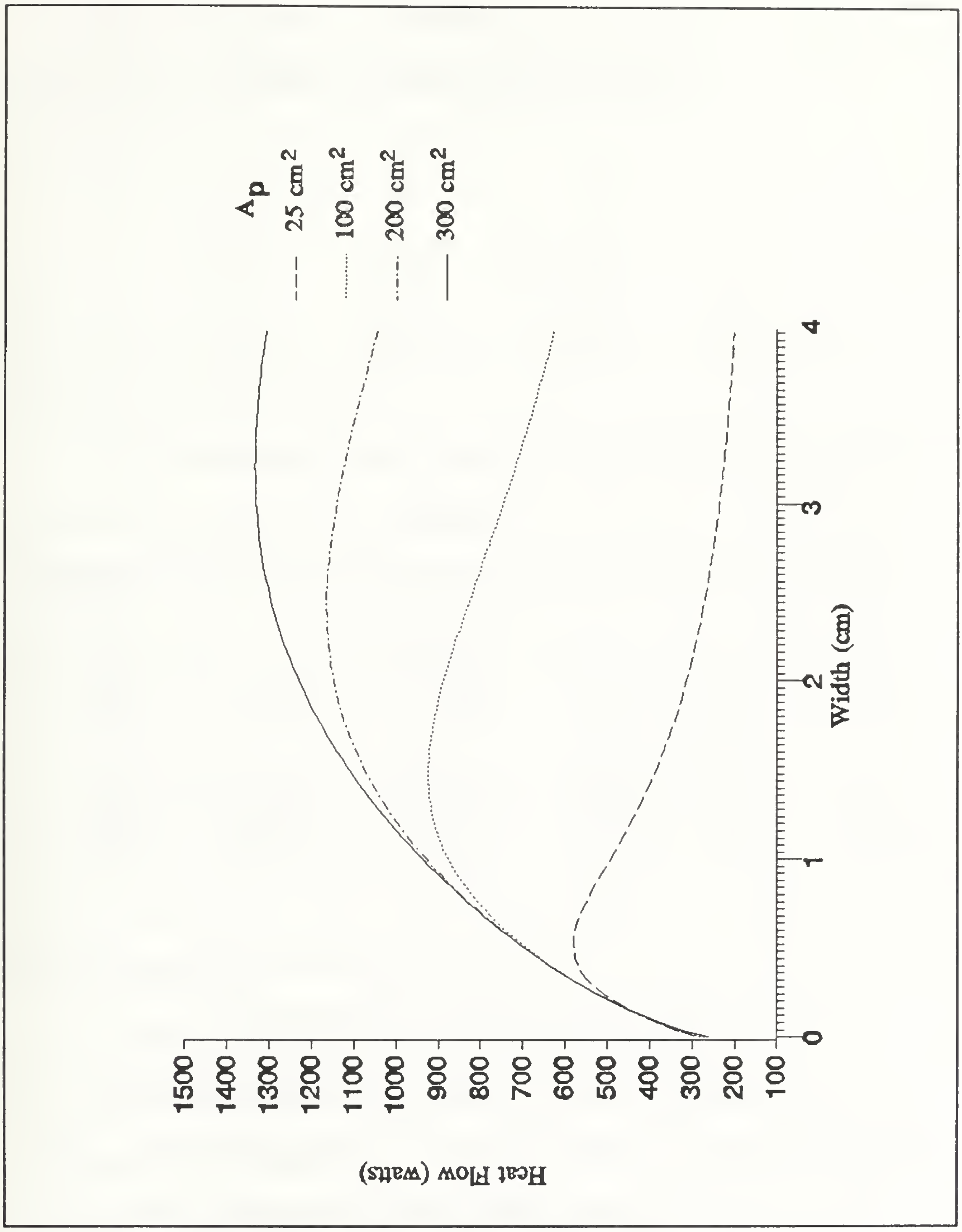

FIGURE 4.3 Profile Area Analysis

TABLE 7

PROFILE AREA ANALYSIS POINTS

A_p (cm^2)	δ_b (cm)	b (m)	q_b (watts)
25	0.334	0.748	507.4
100	0.490	2.041	633.7
200	0.724	2.762	771.2
300	0.334	8.982	523.1

produced from the PROFILE AREA Analysis Program appear in Table 8.

TABLE 8

OPTIMUM GEOMETRY POINTS FOR THE RECTANGULAR PROFILE

A_p (cm^2)	δ_{bo} (cm)	b_o (m)	Mass_o (kg)	q_{bo} (watts)	q_{bo}/m (w/kg)
25	0.646	0.387	6.50	579.76	89.19
100	1.543	0.648	26.00	924.53	35.56
200	2.440	0.819	52.00	1165.83	22.42
300	3.103	0.967	80.00	1334.40	16.68

3. Example Three: Comparison of Various Profiles

A longitudinal fin 1.0 meter deep having a profile area A_p equal to 50 cm^2 is made of aluminum ($k = 209.4$ watt / m K, $\rho = 2.6$ kg / dm^3, and $\epsilon = 0.85$). The base temperature is held at 400 K. Compare the relative performance between the longitudinal fins of rectangular, trapezoidal, and triangular profiles.

The same approach is used as in the previous example. A longitudinal fin of trapezoidal profile having a taper ratio TR = 0.5 is used. The taper ratio is defined as

$$TR = \frac{\delta_a}{\delta_b} \tag{4.3}$$

Figure 4.4 provides a graphical representation which shows the relationship between the three profiles being considered.

The curve in Figure 4.4 produced by the longitudinal fin of trapezoidal profile is midway between the curves produced by the longitudinal fins of rectangular and triangular profiles. Of more importance is the fact that at the location of the optimum geometry points, the longitudinal fin of triangular profile produces the largest heat flow while the longitudinal fin of rectangular profile produces the smallest heat flow. The optimum geometry points extrapolated from the disk file generated from the PROFILE AREA Analysis Program appear in Table 9. Not only is there an optimum fin geometry for a particular profile area A_p but there is also an optimum profile shape which produces maximum heat flow.

4. Example Four: Optimum Geometry Comparison

A longitudinal fin 2.0 meters deep ($\epsilon = 0.95$) is required to remove at least 1000 watts from a spacecraft in a free space environment (V=0). The base temperature is held at 400 K. Using the list of materials appearing in Table 10

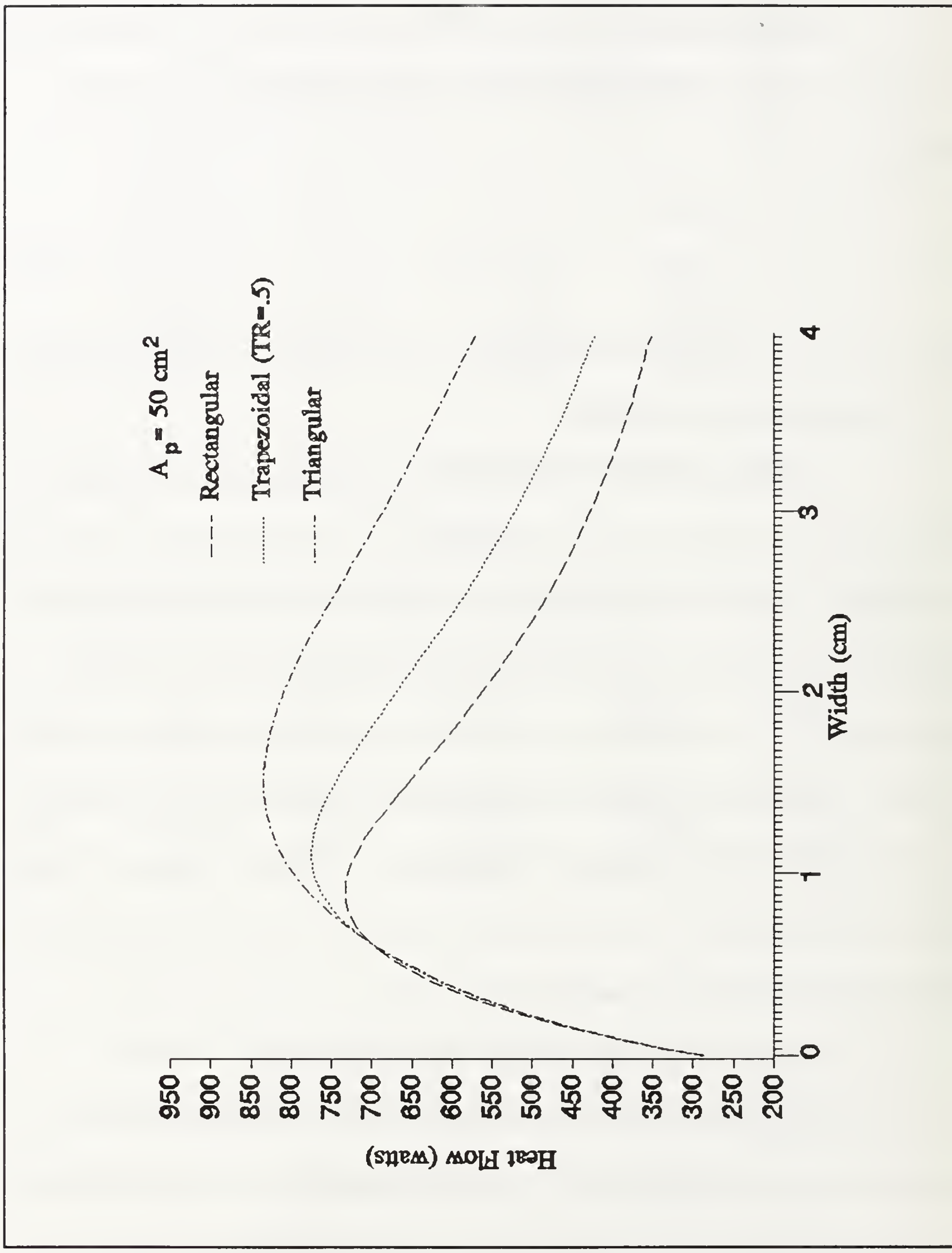

Figure 4.4 Comparisons of Various Profiles

TABLE 9

OPTIMUM GEOMETRY POINTS FOR THE VARIOUS PROFILES

Profile	Taper Ratio	δ_b (cm)	b (m)	Mass (kg)	q_b (watts)	q_b/m (W/kg)
RECT	1.0	0.958	0.522	13.00	732.47	56.343
TRAP	0.5	1.270	0.524	13.00	772.93	59.456
TRI	0.0	1.309	0.763	13.00	826.78	63.598

conduct a heat flow per unit mass comparison using the longitudinal fins of rectangular and triangular profiles.

TABLE 10

MATERIALS USED IN THE OPTIMUM COMPARISON[8]

Material	Chemical Symbol	ρ kg/dm^3	k w / m K
Aluminum	Al	2.6	209.4
Copper	Cu	8.8	384.0
Iron	Fe	7.2	58.0
Magnesium	Mg	1.7	157.0
Molybdenum	Mo	10.2	145.0
Silicon	Si	2.3	83.0
Silver	Ag	10.5	407.0
Zinc	Zn	6.8	140.0

The OPTIMIZE Analysis Program is used for the analysis. The parameters were entered individually into the OPTIMIZE Analysis Program and the results are listed in Table 11.

TABLE 11

OPTIMUM COMPARISON RESULTS

Material	δ_{bo} cm	A_{po_2} cm^2	Mass$_o$ kg	q_{bo} watts	q_{bo}/m w / kg
Ag RECT	0.40	21.73	45.63	997.86	21.86
Ag TRI	0.54	15.09	31.69	995.45	31.41
Al RECT	0.79	42.24	21.96	998.96	45.49
Al TRI	1.05	29.33	15.25	995.47	65.27
Cu RECT	0.43	23.03	40.54	997.94	24.61
Cu TRI	0.57	15.99	28.15	995.45	35.36
Fe RECT	2.84	152.49	221.11	1003.99	4.54
Fe TRI	3.79	105.89	153.54	995.75	6.48
Mg RECT	1.05	56.33	19.60	999.74	51.01
Mg TRI	1.40	39.12	13.61	995.49	73.14
Mo RECT	1.14	61.00	124.43	1000.00	8.04
Mo TRI	1.51	42.36	86.41	995.54	11.52
Si RECT	1.99	106.56	49.66	998.41	20.11
Si TRI	2.65	73.99	34.48	995.59	28.87
Zn RECT	1.18	63.17	85.92	1000.12	11.64
Zn TRI	1.57	43.87	59.66	995.50	16.68

Figure 4.5 shows a bar graph representation of the heat flow per unit mass relationship using the various materials listed in Table 10. It is evident from Figure 4.5 that the optimized longitudinal fin of triangular profile made of magnesium is the most efficient material considered of those listed in Table 10, with aluminum being a close second. The melting temperature of both materials ($\approx$ 900 K) is far above the base temperature (400 K) stated in the problem statement. Comparing Tables 10 and 11 materials having a small value for the density ($\rho < 3.0$ kg / dm^3) and a large value for the thermal conductivity ($k > 150$ watt / m K) produce large values for heat flow per unit mass.

C. DERIVATION OF WORKING OPTIMUM GEOMETRY EQUATIONS

As shown in Tables 8 and 9 optimum geometry points can be determined using the PROFILE AREA Analysis Program. The overall success of finding the dimensions for the optimum geometry using the PROFILE AREA Analysis Program has several limitations. The first limitation is that the starting and ending fin base width values must be known. And it can be argued that a second limitation is that excessive analysis program run times which might exceed ten minutes.

Relationships for the optimum geometry for the longitudinal fins of rectangular, trapezoidal, and triangular profiles would remove both limitations. Kern and Kraus[9] and Chung and Nguyen[10] stated that the dimensions for optimum fin geometry can be expressed by

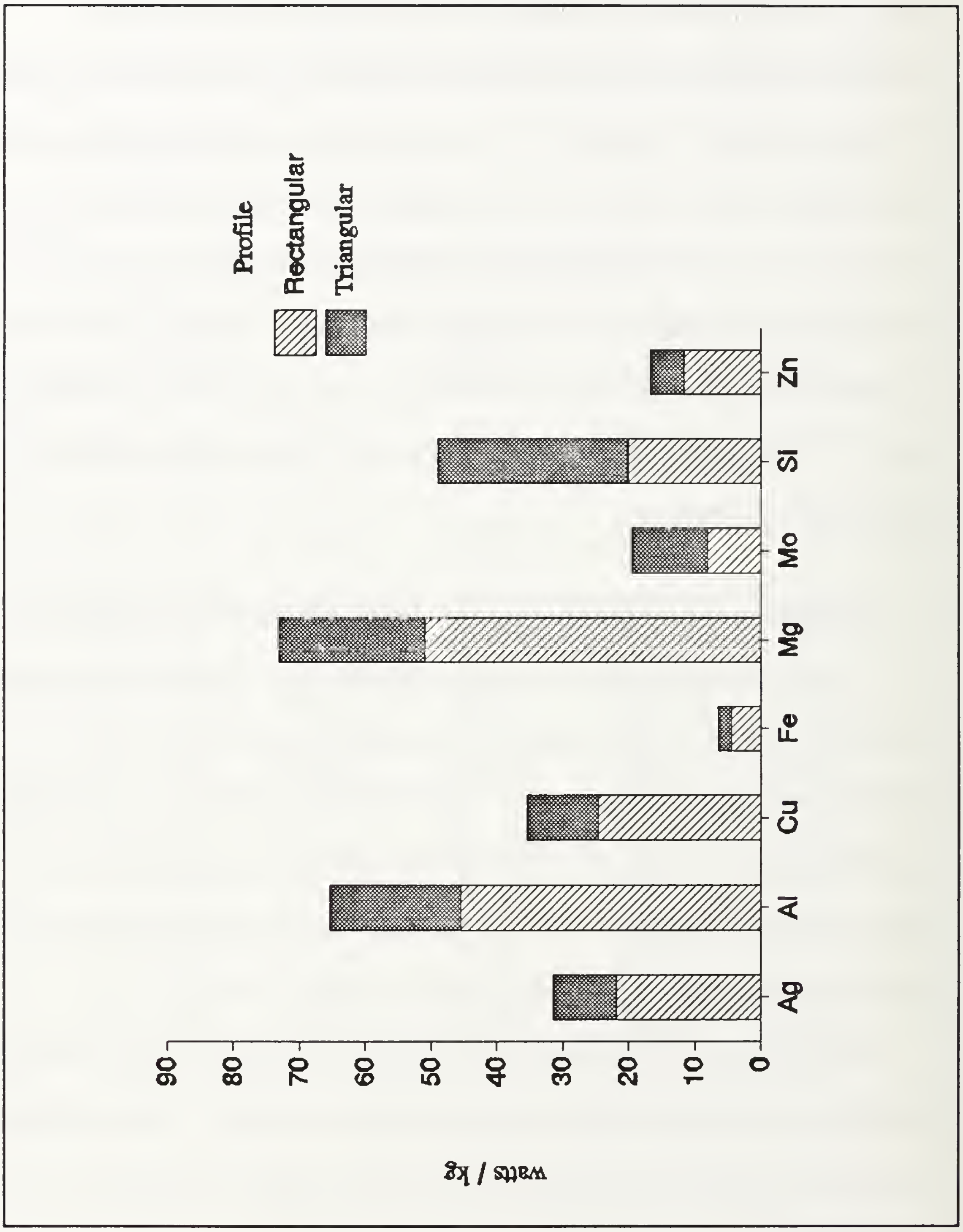

Figure 4.5 Optimum Profile Comparison Using Various Materials

$$b_o = P_1 \frac{q_b}{L \sigma \epsilon T_b^4} \tag{4.4}$$

$$\delta_{b_o} = P_2 \frac{1}{k \sigma \epsilon T_b^5} \left(\frac{q_b}{L}\right)^2 \tag{4.5}$$

$$Ap_o = P_3 \frac{1}{k \sigma^2 \epsilon^2 T_b^9} \left(\frac{q_b}{L}\right)^3 \tag{4.6}$$

where $P_1(TR)$, $P_2(TR)$, and $P_3(TR)$ are defined as polynomials in which the taper ratio TR is the independent variable.

In general the polynomial P is expressed in the form of

$$P(TR) = C_1 \, TR^{\,n-1} + C_2 \, TR^{\,n-2} + \dots + C_n \, TR^{\,0} \tag{4.7}$$

where n is the degree of the polynomial.

The polynomials P_1, P_2, and P_3 are determined from running the PROFILE AREA Analysis Program using a slightly modified code. The computer code listing appears in Appendix B. A random number generator is used to generate the input parameters. The random number generator produces input parameters in the range listed in Table 12. Murray-Gardner (Table 1) assumption number five (the fin width is small compared with its depth and length) is checked in order to insure that the original assumptions are kept valid.

TABLE 12

RANGE OF THE RANDOM INPUT PARAMETERS

$0.01 \leq \epsilon \leq 0.99$
$20 \leq L\ (\mathrm{cm}) \leq 220$
$10 \leq k\ (\mathrm{watt\ /\ m\ K}) \leq 210$
$100 \leq T_b\ (\mathrm{K}) \leq 600$
$25 \leq q_b\ (\mathrm{watts}) \leq 1025$
$0.01 \leq$ Taper Ratio TR ≤ 0.99

The program enters into a loop calculating the heat flow q_b using the random input parameters. The program remains in this loop until the maximum heat flow q_b is found. Once found the program writes the random input parameters and the value determined to be the maximum heat flow q_b to a disk file. Another set of random input parameters is generated and the maximum heat flow q_b once again is calculated. This sequence continues until 100 optimum data sets are written to a disk file. Three data sets using the random input parameters are produced, one each for the longitudinal fins of rectangular, trapezoidal, and triangular profiles.

The last step in determining the polynomials P_1, P_2, and P_3 is to fit the collected data stored in the disk file to a "best" polynomial fit having the smallest degree d using the least square principle. Equations (4.4) through (4.6) are rewritten as

$$P(TR)_1 = \frac{L\, b_o\, \sigma\, \epsilon\, T_b^4}{q_b} \tag{4.8}$$

$$P(TR)_2 = \delta_{b_o}\, k\, \sigma\, \epsilon\, T_b^5 \left(\frac{L}{q_b}\right)^2 \tag{4.9}$$

$$P(TR)_3 = Ap_o\, k\, \sigma^2\, \epsilon^2\, T_b^9 \left(\frac{L}{q_b}\right)^3 \tag{4.10}$$

Equations (4.8) through (4.10) are placed into a MATLAB "m" file. Each of the collected data files are read into MATLAB and were evaluated using the MATLAB polynomial fitting function polyfit with the taper ratio TR as the independent variable using varying degrees d in order to find the polynomials P_1, P_2, and P_3 which "best" approximate the previously collected data from the disk file. The results appear in Table 13. The listing of the MATLAB evaluation m files appear in Appendix C.

The standard error ε_s is defined as

$$\varepsilon_s = \left[\frac{\sum (y_i - \hat{y}_i)^2}{N - 1}\right]^{1/2} \tag{4.11}$$

where $\hat{y}_i$ is the computed value of y_i and there are N values.

TABLE 13

COEFFICIENTS OF THE OPTIMUM POLYNOMIALS

Profile	Degree d	C_1	C_2	C_3	Standard Error
P1 RECT	2	0.8675	0.0000	0.0000	3.54e-3
P1 TRAP	2	-0.0290	0.8138	0.0000	2.12e-3
P1 TRI	2	0.0000	0.9040	0.0000	2.55e-3
P2 RECT	2	1.8648	0.0000	0.0000	1.63e-2
P2 TRAP	3	0.3881	-1.0324	2.7522	1.22e-2
P2 TRI	2	0.0000	2.4852	0.0000	1.40e-2
P3 RECT	2	1.6178	0.0000	0.0000	2.06e-2
P3 TRAP	2	0.5147	1.1382	0.0000	1.24e-2
P3 TRI	2	0.0000	1.1234	0.0000	9.46e-3

The equation for the optimum profile area A_{po} for either the rectangular, triangular, or trapezoidal profiles is written as

$$A_{po} = (1.1382 + 0.5147\ TR) \left[\frac{1}{k\ \sigma^2\ \epsilon^2} \left(\frac{q_b}{L\ T^3{}_b} \right)^3 \right] \tag{4.12}$$

V. CONCLUSIONS

This thesis has described a new technique that uses a parameterization called the thermal transmission matrix to represent a single fin to evaluate the performance of several fins. The algorithm is based on the simple ideal of subdividing the fin into 100 subfins and then the thermal transmission matrix is calculated for each subfin. Then, by a simple matrix multiplication of the individual subfin thermal transmission matrices, the overall thermal transmission matrix is computed. Not only will this produce the base and tip conditions of the overall fin entity, but the temperature distribution and heat flow at any point between the fin base and the fin tip can be easily determined.

Several conclusions can be made from the Chapter 4 Program Analysis Results and Discussion. It was observed that the profile area A_p, increases as the cube of the heat flow q_b. In order to double the heat flow q_b, the profile area A_p must be increased eight times.

Material selection is very important. The weight of the fin is proportional to the density and the thermal conductivity is inversely proportional to the profile area. Materials having a thermal conductivity greater than 150 watt / m K and a density less than 3 kg / dm^3 are generally the best suited materials. For the same material, environmental conditions, heat flow, and base temperature, the optimum longitudinal fin of triangular profile requires only 69 percent as much material as the optimum longitudinal fin of rectangular profile, with the optimum longitudinal fin of trapezoidal profile falling in between.

Simple working relationships to compute the optimum fin dimensions are found. Table 16 compares the findings from this thesis with the results from previous work by others. The differences between the optimum dimension polynomial coefficients listed in Table 16 are very small, which verifies the accuracy of the thermal transmission matrix parameterization technique.

TABLE 16

OPTIMUM DIMENSION POLYNOMIAL COEFFICIENTS

	P1	P2	P3	
RECTANGULAR				
	.8675	1.8648	1.6178	This Thesis
	.8846	1.8473	1.6341	Chung and Nguyen[11]
	.8842	1.8579	1.6056	Liu[12]
	.8846	1.8473	1.6341	Bartas and Sellers[13]
TRIANGULAR				
	.9040	2.4852	1.1234	This Thesis
	.9485	2.3104	1.0957	Chung and Nguyen
	.9546	2.3027	1.0991	Nilson and Curry[14]
	.9091	2.4255	1.1025	Kern and Kraus[15]

This new approach to fin synthesis and analysis permits an exact solution to any degree of accuracy that the user wishes to specify and produces the result in a computational efficient manner.

APPENDIX A COMPUTER CODE LISTING

```
/* ******************************************************************************

      Main Program

****************************************************************************** */

#include <process.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#define Yes  1
#define No   0

char   *S1[] = { " watt / deg C", " BTU / hr deg F" };
char   *S2[] = { " m ^ 2", " cm ^ 2", " ft ^ 2", " in ^ 2", " mm ^ 2" };
char   *S3[] = { " watts", " kw", " BTU / sec", " BTU / hr" };
char   *S4[] = { " deg C / watt", " deg F hr / BTU" };
char   *S5[] = { " m", " cm", " ft", " in", " mm" };
char   *S6[] = { " kg", " grams", " lbm", " oz"  };
char   *S7[] = { " deg K", " deg C", " deg F", " deg R" };
char   *S8[] = { " watt / m deg K", " BTU / ft hr deg F" };
char   *S9[] = { " No", " Yes"};

char   *O[]  = {    "<1> watt / deg C  <2> BTU / hr deg F",
                    "<1> m ^ 2  <2> cm ^ 2  <3> ft ^ 2  <4> in ^ 2  <5> mm ^ 2",
                    "<1> watts  <2> kw  <3> BTU / sec  <4> BTU / hr",
                    "<1> deg C / watt  <2> deg F hr / BTU",
                    "<1> m  <2> cm  <3> ft  <4> in  <5> mm",
                    "<1> kg  <2> grams  <3> lbm  <4> oz",
                    "<1> deg K  <2> deg C  <3> deg F  <4> deg R",
                    "<1> watt / m deg K  <2> BTU / ft hr deg F",
                    "<1> No  <2> Yes" };

void print_main_menu(), print_menu_one(), print_menu_two();
void setup(), overlays(), update_units();

float error=0.1;
int T=0, L=0, M=0, S=0, H=0, d=4, K=0, A=1, Z=0, Y=0;
char main_key=0;

int main(){

FILE *data_in;

if (  (data_in = fopen("units.ext","r"))  != NULL){
      fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&A,&Z,&Y,&error);
      fclose(data_in); }

print_main_menu();

do{
V     if (main_key == '1') setup();

      if (main_key >= '2' && main_key <= '4') overlays(); }

while (( main_key = getch() ) != '5');
window(1,1,80,25);
clrscr();
_setcursortype(_NORMALCURSOR);
```

```
return 0;}

void overlays(void){
char key=0;
int Quit = No, result;

print_menu_one();

do
      switch (key){

      case 'A': clrscr(); Quit = Yes;
            if (main_key == '2')  result = spawnl(P_WAIT, "RECT.EXE", NULL);
            if (main_key == '3')  result = spawnl(P_WAIT, "TRAP.EXE", NULL);
            if (main_key == '4')  result = spawnl(P_WAIT, "TRI.EXE", NULL);
            break;

      case 'B': clrscr(); Quit = Yes;
            if (main_key == '2')  result = spawnl(P_WAIT, "ARECT.EXE", NULL);
            if (main_key == '3')  result = spawnl(P_WAIT, "ATRAP.EXE", NULL);
            if (main_key == '4')  result = spawnl(P_WAIT, "ATRI.EXE", NULL);
            break;

      case 'C': clrscr(); Quit = Yes;
            if (main_key == '2')  result = spawnl(P_WAIT, "ORECT.EXE", NULL);
            if (main_key == '3')  result = spawnl(P_WAIT, "OTRAP.EXE", NULL);
            if (main_key == '4')  result = spawnl(P_WAIT, "OTRI.EXE", NULL);}

while (( key = toupper(getch()) ) != 'D' && Quit == No);

print_main_menu();
return;}

void setup(void){
char key=0;

print_menu_two();

do
      switch (key){

      case 'A':     gotoxy(2,17); clreol(); cprintf("%s",O[0]);
                    while (key != '1' && key != '2') key = getch();
                    Y = key - '1';print_menu_two(); break;

      case 'B':     gotoxy(1,17); clreol(); cprintf("%s",O[1]);
                    while (key != '1' && key != '2'&& key != '3'&& key != '4'&& key != '5')
                    key = getch();
                    A = key - '1';print_menu_two(); break;

      case 'C':     gotoxy(1,17); clreol(); cprintf("%s",O[2]);
                    while (key != '1' && key != '2'&& key != '3'&& key != '4') key = getch();
                    H = key - '1';print_menu_two(); break;

      case 'D':     gotoxy(2,17); clreol(); cprintf("%s",O[3]);
                    while (key != '1' && key != '2') key = getch();
                    Z = key - '1';print_menu_two(); break;

      case 'E':     gotoxy(2,17); clreol(); cprintf("%s",O[4]);
                    while (key != '1' && key != '2'&& key != '3'&& key != '4'&& key != '5')
                    key = getch();
                    L = key - '1';print_menu_two(); break;

      case 'F':     gotoxy(3,17); clreol(); cprintf("%s",O[5]);
                    while (key != '1' && key != '2'&& key != '3'&& key != '4') key = getch();
                    M = key - '1';print_menu_two(); break;

      case 'G':     gotoxy(1,17); clreol(); cprintf("%s",O[6]);
                    while (key != '1' && key != '2'&& key != '3'&& key != '4') key = getch();
                    T = key - '1';print_menu_two(); break;

        case 'H':   gotoxy(1,17); clreol(); cprintf("%s",O[7]);
                    while (key != '1' && key != '2') key = getch();
                    K = key - '1';print_menu_two(); break;

        case 'I':   gotoxy(2,17); clreol(); cprintf("%s",O[4]);
                    while (key != '1' && key != '2'&& key != '3'&& key != '4'&& key != '5')
                    key = getch();
                    d = key - '1';print_menu_two(); break;

        case 'J':   gotoxy(12,17); clreol(); cprintf("%s",O[8]);
                    while (key != '1' && key != '2') key = getch();
                    S = key - '1';print_menu_two(); break;

        case 'K':   gotoxy(22,12); clreol(); scanf("%f",&error);
                    print_menu_two();}

while (( key = toupper(getch()) ) != 'L');

update_units();
print_main_menu();
return;}

void update_units(void){

FILE *unit;
unit = fopen("units.ext","w");
fprintf(unit,"%d %d %d %d %d %d %d %d %d %d %f",T,L,M,S,H,d,K,A,Z,Y,error);
fclose(unit);
return;}

void print_main_menu(void){

_setcursortype(_NOCURSOR);
window(1,1,80,25);
clrscr();
gotoxy(27,3); cprintf("EXTENDED SURFACE ANALYSIS");
window(20,6,80,25);

cprintf("<1>  Setup / Initialization\r\n");
cprintf("<2>  Fin of Rectangular Profile\r\n");
cprintf("<3>  Fin of Trapezoidal Profile\r\n");
cprintf("<4>  Fin of Triangular Profile\r\n");
cprintf("<5>  Return to DOS");
return;}

void print_menu_one(void){

clrscr();
gotoxy(1,1);
cprintf("------------------------------------\r\n");
cprintf("    <A>  Predict the Performance\r\n");
cprintf("    <B>  Analyze the Profile Area\r\n");
cprintf("    <C>  Optimize the Dimensions\r\n");
cprintf("    <D>  Return to Main Menu\r\n");
cprintf("------------------------------------");
return;}

void print_menu_two(void){

clrscr();
gotoxy(1,1);
cprintf("-----------------------------------------\r\n");
cprintf("<A>  Admittance   : %s\r\n", S1[Y]);
cprintf("<B>  Area         : %s\r\n", S2[A]);
cprintf("<C>  Heat         : %s\r\n", S3[H]);
cprintf("<D>  Impedance    : %s\r\n", S4[Z]);
cprintf("<E>  Depth        : %s\r\n", S5[L]);
cprintf("<F>  Mass          : %s\r\n", S6[M]);
cprintf("<G>  Temperature   : %s\r\n", S7[T]);
cprintf("<H>  Thermal K     : %s\r\n", S8[K]);
cprintf("<I>  Width         : %s\r\n", S5[d]);
cprintf("<J>  Save Data     : %s\r\n", S9[S]);
cprintf("<K>  Percent Error:  %f\r\n", error);
cprintf("<L>  Return to Main Menu\r\n");
cprintf("----------------------------------------");
return;}
```

```
/* ***************************************************************************

      PERFORMANCE Analysis Program

      Longitudinal Fin of Rectangular Profile

*************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"

#define      Yes    1
#define      No     0
#define      N      100
#define      sigma  5.66961e-8

char         file_name[20];
double       A_(), B_(), C_(), D_(), m(), hr(), Yb(), A, B, C, D;
double       length, width, depth;
double       inc, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double       qb=0,A1, B1, C1, D1, Ta, Tb, Ttip, Tave, temp, qa, segment_diff;
float        error, V, epsilon, k, Tbase, density, mass, K2, eta, PN;
int          T, L, M, S, H, d, K, i, X, Z, Y;
int          overflow, possible_overflow = No, number_of_tries=0;
void         input(), compute(), results();

/* *************************************************************************

      MAIN PROGRAM

************************************************************************* */

int main(){

FILE *data_in, *out;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = .0001 + Tave;

inc = .5 * Ttip; i = 1;

while (fabs(overall_diff) >  (Tbase * error)  || i <= N ){

      overflow = No;      i = 1;       Ta = Ttip; Tb = Ta;
      qa = sigma * width * epsilon * pow(Ta, 4) * depth;

      while (i <= N && overflow == No){

            segment_diff = 10;

            while (fabs(segment_diff) > ((Tb * error) / N) && overflow == No){

                  A = A_(); B = B_(); C = C_(); D = D_();
                  temp = A * Ta + B * qa;
```

```
                qb   = C * Ta + D * qa;
                segment_diff = ( temp - Tb) / 2;
                Tb = temp + segment_diff;

                if (Tb > Tbase + 50 || Tb < 0){
                      overflow   = Yes; possible_overflow = Yes;
                      if (Ttip < Tbv_min ) Tbv_min = Ttip;
                      inc = (Tbv_min - Tmax) / 2;
                      Ttip = fabs(Tbv_min - inc);
                      if (Ttip < Tave) Ttip = Tave;}}

          if (overflow == No){
                data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                Ta = Tb; qa = qb; ++i;}}

     if (number_of_tries > 29){
          clrscr(); cprintf("Fatal Error:  Unable to Compute");
          getch(); exit(0);}

     if (overflow == No){
          if (Tmax < Ttip)
                if ( Tb < Tbase ) Tmax = Ttip;

          overall_diff = (Tb - Tbase) / 2;

          if ( possible_overflow == Yes){
                      inc = inc / 2;
                      if (overall_diff > 0 )  Ttip = Ttip - inc;
                      else  Ttip = Ttip + inc;}
          else
                Ttip = Ttip - overall_diff;

          number_of_tries++;
          if (Ttip < Tave) Ttip = Tave;
          gotoxy(27,24);
          cprintf("N = %d Ta = %4.1f Tb = %4.1f   ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));}}

for (i = N - 1; i >= 1; i--){
     A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
     C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
     A = A1; B = B1; C=C1; D=D1;}

if (S == 1){
     out = fopen(file_name, "w");
     for (i = 1; i <= N; i++)
          fprintf(out, "%e,  %e,  %e\n", w_L(d, i * length / N), w_T(T,data[i][1]), w_H(H,
                data[i][0]) );
     fclose(out);}

Ta = Ttip;
mass = width * length * depth * density * 1000;
PN = length * sqrt( sigma * epsilon * pow(Tbase,3) / ( k * width));
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase,4));
results();
exit(0);
return;}

/* **************************************************************************

     OUTPUT PROCEDURE

************************************************************************** */

void results(void){

clrscr();
cprintf("\a          THE LONGITUDINAL FIN OF RECTANGULAR PROFILE");
cprintf("\r\n    ================================================================\r\n");
cprintf("                          Density = %4.2f kg / dm^3\r\n",density);
cprintf("                        Fin Depth = "); p_L(L, depth);
cprintf("                       Total Mass = "); p_M(M, mass);
cprintf("                       Fin Length = "); p_L(L, length);
cprintf("                       Fin Width  = "); p_L(d, width);
cprintf("                       Emissivity = %4.2f\r\n", epsilon);
cprintf("                     Profile Area = "); p_A(X, width*length);
cprintf("                   Profile Number = %4.3f\r\n",PN);
cprintf("                   Base Heat Flow = "); p_H(H, qb);
cprintf("                   Fin Efficiency = %4.3f\r\n",eta);
cprintf("                 Base Temperature = "); p_T(T,Tb);
cprintf("                 Edge Temperature = "); p_T(T,Ta);
cprintf("             Thermal Conductivity = "); p_K(K, k);
cprintf("          Environmental Parameter = %4.2f\r\n", V);
cprintf("      ==================================================================\r\n");
cprintf("                   Transmission Parameter Matrix \r\n\r\n\r\n");
cprintf("      A = %4.4f                   B = ", A); p_Z(Z,B);
cprintf("      C = "); p_Y(Y,C);
cprintf("       D = %4.4f", D);
getch();
return;}

/* ************************************************************************

      INPUT PROCEDURE

************************************************************************ */

void input(void){

clrscr();_setcursortype(_NORMALCURSOR);
cprintf("\r\n          THE LONGITUDINAL FIN OF RECTANGULAR PROFILE\n\r");
cprintf("\r\n    ==================================================================\r\n");
cprintf("                              Fin Depth "); depth = i_L(L);
cprintf("                             Fin Length "); length = i_L(L);
cprintf("                             Fin Width  "); width = i_L(d);
cprintf("                                  Emissivity = "); scanf("%f", &epsilon);
cprintf("                       Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                    Base Temperature "); Tbase = i_T(T);
cprintf("      Thermal Conductivity "); k = i_K(K);
cprintf("           Environmental Parameter (0 to .8) = "); scanf("%f", &V);

if (S == 1) {
      cprintf("                          Output Filename is = ");
      scanf("%s", file_name);}

cprintf("\r\n==================================================================");

if ( epsilon > 1.0 || epsilon <= 0){
      cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
      getch(); exit(0);}

if ( V < 0 || V > 0.8){
      cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
      getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* ************************************************************************

      GLOBAL FUNCTIONS

************************************************************************ */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}
```

```
double m(void){
return( sqrt( ( 2.0 * hr()) / ( k * width) ) );}

double Yb(void){
return( sqrt( 2.0 * hr() * k * width) * depth );}

double A_(void){
return( cosh( m() * length / N )  );}

double B_(void){
return ( sinh( m() * length / N ) / Yb() );}

double C_(void){
return ( sinh( m() * length / N ) * Yb() );}

double D_(void){
return( cosh( m() * length / N )  );}
```

```
/* ****************************************************************************

      PERFORMANCE Analysis Program

      Longitudinal Fin of Trapezoidal Profile

**************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "bessel.h"
#include "units.h"

#define      Yes    1
#define      No     0
#define      N      100
#define      sigma  5.66961e-8

char         file_name[20];
double       A_(), B_(), C_(), D_(), m(), hr(), u(double x), A, B, C, D;
double       length, depth, edge_width, base_width, fin_taper;
double       b1, db, a, b, ua, ub, Ta, Tb, a_width, b_width;
double       qb=0, inc, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double       A1, B1, C1, D1, taper_ratio, Ttip, temp, qa, segment_diff;
float        V, epsilon, k, error, Tbase, density, mass, K2, Ap, eta, PN;
int          T, L, M, S, H, d, K, i, X, Z, Y;
int          overflow, possible_overflow = No, number_of_tries=0;
void         input(), results();

/* *************************************************************************

      MAIN PROGRAM

************************************************************************** */

int main(){

FILE *data_in, *out;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

taper_ratio = edge_width / base_width;
b1 = length / ( 1 - taper_ratio );
fin_taper = atan2( base_width, 2*b1 );
db = length / N;

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > (Tbase * error)   || i <= N ){


      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      qa = sigma * edge_width * epsilon * pow(Ta, 4) * length;
      a = b1 - length;
```

```
        while (i <= N && overflow == No){

                segment_diff = 10;
                a_width = edge_width * a / ( b1 - length);
                b_width = edge_width * ( a + db ) / ( b1 - length);

                while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                        ua = u(a) ; ub = u(a + db);
                        A = A_(); B = B_(); C = C_(); D = D_();
                        temp = A * Ta + B * qa;
                        qb   = C * Ta + D * qa;
                        segment_diff = ( temp - Tb) / 2;
                  .     Tb = temp + segment_diff;

                        if (Tb > Tbase + 50 || Tb < 0){
                                overflow   = Yes; possible_overflow = Yes;
                                if (Ttip < Tbv_min ) Tbv_min = Ttip;
                                inc = (Tbv_min - Tmax) / 2;
                                Ttip = fabs(Tbv_min - inc);
                                if (Ttip < Tave) Ttip = Tave;}}

                if (overflow == No){
                        data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                        data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                        Ta = Tb; qa = qb; a = a + db; ++i;}}

        if (number_of_tries > 29){
                clrscr(); cprintf("Fatal Error:  Unable to Compute");
                getch(); exit(0);}

        if (overflow == No){
                if (Tmax < Ttip)
                        if (Tb < Tbase) Tmax = Ttip;

                overall_diff = (Tb - Tbase) / 2;

                if ( possible_overflow == Yes){
                                inc = inc / 2;
                                if (overall_diff > 0 )  Ttip = Ttip - inc;
                                else  Ttip = Ttip + inc;}
                else
                        Ttip = Ttip - overall_diff;
                number_of_tries++;
                gotoxy(27,24);
                cprintf("N = %d Ta = %4.1f Tb = %4.1f    ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
                if (Ttip < Tave) Ttip = Tave;}}

for (i = N - 1; i >= 1; i--){
        A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
        C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
        A = A1; B = B1; C=C1; D=D1;}

if (S == 1){
        out = fopen(file_name, "w");
        for (i = 1; i <= N; i++)
                fprintf(out, "%e,  %e,  %e\n", w_L(d, i * length / N), w_T(T,data[i][1]), w_H(H,
                        data[i][0]) );
        fclose(out);}

Ta = Ttip;
Ap = .5 * length * ( edge_width + base_width );
mass = Ap * length * density * 1000;
PN = 2 * sigma * epsilon * pow(length, 2) * pow(Tbase,3) / (k* base_width);
eta = qb / ( 2 * sigma * epsilon * length * length * pow(Tbase,4));
results();
exit(0);
return;}
```

```
/* ***************************************************************************

       OUTPUT PROCEDURE

*************************************************************************** */

void results(void){

clrscr();
cprintf("          THE LONGITUDINAL FIN OF TRAPEZOIDAL PROFILE");
cprintf("\r\n    ==================================================================\r\n");
cprintf("                         Density = %4.2f kg / dm^3\r\n",density);
cprintf("                       Fin Depth = "); p_L(L, depth);
cprintf("                      Total Mass = "); p_M(M, mass);
cprintf("                      Fin Length = "); p_L(L, length);
cprintf("                      Emissivity = %4.2f\r\n", epsilon);
cprintf("                    Profile Area = "); p_A(X,Ap);
cprintf("                  Profile Number = %4.3f\r\n",PN);
cprintf("                  Base Heat Flow = "); p_H(H, qb);
cprintf("                  Fin Efficiency = %4.3f\r\n",eta);
cprintf("                 Fin Edge Width  = "); p_L(d, edge_width);
cprintf("                 Fin Base Width  = "); p_L(d, base_width);
cprintf("                Base Temperature = "); p_T(T,Tb);
cprintf("                Edge Temperature = "); p_T(T,Ta);
cprintf("            Thermal Conductivity = "); p_K(K, k);
cprintf("         Environmental Parameter = %4.2f\r\n", V);
cprintf("    ==================================================================\r\n");
cprintf("                  Transmission Parameter Matrix \r\n\r\n");
cprintf("    A = %4.4f                    B = ", A); p_Z(Z,B);
cprintf("    C = "); p_Y(Y,C);
cprintf("     D = %4.4f", D);
getch();
return;}

/* ***************************************************************************

       INPUT PROCEDURE

*************************************************************************** */

void input(void){

clrscr(); _setcursortype(_NORMALCURSOR);
cprintf("\r\n          THE LONGITUDINAL FIN OF TRAPEZOIDAL PROFILE\r\n");
cprintf("\r\n    ==================================================================\r\n");
cprintf("                              Fin Depth "); depth = i_L(L);
cprintf("                             Fin Length "); length = i_L(L);
cprintf("                          Fin Edge Width  "); edge_width = i_L(d);
cprintf("                          Fin Base Width  "); base_width = i_L(d);
cprintf("                               Emissivity = "); scanf("%f", &epsilon);
cprintf("                       Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                     Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("         Environmental Parameter (0 to .8) = "); scanf("%f", &V);

if (S == 1) {
      cprintf("                           Output Filename is = ");
      scanf("%s", file_name);}

cprintf("\r\n    ==================================================================");

if ( edge_width >= base_width){
      cprintf("\r\n\r\nFATAL ERROR:  Fin Edge Width >= Fin Base Width");
      getch(); exit(0);}

if ( epsilon > 1.0 || epsilon <= 0){
      cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
      getch(); exit(0);}
if ( V < 0 || V > 0.8){
      cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
      getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* **************************************************************************

      GLOBAL FUNCTIONS

************************************************************************** */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( hr() / (k * sin(fin_taper)));}

double u(double x){
return( 2 * sqrt(x * m() ) );}

double A_(void){
return(  ua * (I1(ua)*K0(ub) + I0(ub)*K1(ua))  );}

double B_(void){
return (  ub*ub*(I0(ub)*K0(ua)-I0(ua)*K0(ub)) / ( 2*k*b_width*depth*m()));}

double C_(void){
return ( 2*k*a_width*depth*m()*ub*(I1(ub)*K1(ua) - I1(ua)*K1(ub))/(ua) );}

double D_(void){
return( ub*( I0(ua)*K1(ub)+I1(ub)*K0(ua)) );}
```

```
/* ******************************************************************************

       PERFORMANCE Analysis Program

       Longitudinal Fin of Triangular Profile

****************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "bessel.h"
#include "units.h"

#define       Yes    1
#define       No     0
#define       N      99
#define       sigma  5.66961e-8

char          file_name[20];
double        A_(), B_(), C_(), D_(), m(), hr(), u(double x), A, B, C, D;
double        length, depth, edge_width, base_width, fin_taper;
double        b1, db, a, ua, ub, Ta, Tb, a_width, b_width;
double        qb=0, inc, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double        A1, B1, C1, D1, Ttip, temp, qa, segment_diff;
float         V, epsilon, k, error, Tbase, density, mass, K2, Ap, PN, eta;
int           T, L, M, S, H, d, K, i, X, Z, Y;
int           overflow, possible_overflow = No, number_of_tries=0;
void          input(), results();

/* ***************************************************************************

       MAIN PROGRAM

*************************************************************************** */

int main(){

FILE   *data_in, *out;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

db = length / ( N + 1 );
b1 = length;
length = length - db;
fin_taper = atan2( base_width, 2*b1 );

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
       Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
       Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

       overflow = No; i = 1; Ta = Ttip; Tb = Ta;
       segment_diff = 10;

       while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){
        ub = u(db);
        temp = Ta / I0(ub);
        C    = 2 * k * edge_width * depth * pow(m(),2) * I1(ub) / (ub * I0(ub));
        qb   = C * temp;
        segment_diff = ( temp - Tb) / 2;
        Tb = temp + segment_diff;

        if (Tb > Tbase + 50 || Tb < 0){
                overflow   = Yes; possible_overflow = Yes;
                if (Ttip < Tbv_min ) Tbv_min = Ttip;
                inc = (Tbv_min - Tmax) / 2;
                Ttip = fabs(Tbv_min - inc);
                if (Ttip < Tave) Ttip = Tave;}}

if (overflow == No){
        data[i][0] = qb; data[i][1] = Tb; data[i][2] = 1;
        data[i][3] = 0;  data[i][4] = C;  data[i][5] = 1;
        Ta = Tb; qa = qb; ++i;}

a = b1 - length;

while (i <= N && overflow == No){

        segment_diff = 10;
        a_width = edge_width * a / ( b1 - length);
        b_width = edge_width * ( a + db ) / ( b1 - length);

        while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                ua = u(a) ; ub = u(a + db);
                A = A_(); B = B_(); C = C_(); D = D_();
                temp = A * Ta + B * qa;
                qb   = C * Ta + D * qa;
                segment_diff = ( temp - Tb) / 2;
                Tb = temp + segment_diff;

                if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

        if (overflow == No){
                data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                Ta = Tb; qa = qb; a = a + db; ++i;}}

if (number_of_tries > 29){
        clrscr(); cprintf("Fatal Error:  Unable to Compute");
        getch(); exit(0);}

if (overflow == No){
        if (Tmax < Ttip)
                if (Tb < Tbase) Tmax = Ttip;

        overall_diff = (Tb - Tbase) / 2;

        if ( possible_overflow == Yes){
                        inc = inc / 2;
                        if (overall_diff > 0 )  Ttip = Ttip - inc;
                        else  Ttip = Ttip + inc;}
        else
                Ttip = Ttip - overall_diff;
        number_of_tries++;
        gotoxy(27,24);
        cprintf("N = %d Ta = %4.1f Tb = %4.1f  ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
        if (Ttip < Tave) Ttip = Tave;}}
```

```
for (i = N - 1; i >= 1; i--){
      A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
      C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
      A = A1; B = B1; C=C1; D=D1;}

length = length + db;

if (S == 1){
      out = fopen(file_name, "w");
      for (i = 1; i <= N; i++)
            fprintf(out, "%e,  %e,  %e\n", w_L(d, i * length / N), w_T(T,data[i][1]), w_H(H,
data[i][0]));
      fclose(out);}

Ta = Ttip;
Ap = .5 * length * base_width;
mass = Ap * depth * density * 1000;
PN = 2 * sigma * epsilon * pow(length,2) * pow(Tbase,3) / (k*base_width);
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase,4));
results();
exit(0);
return;}

/* ************************************************************************

      OUTPUT PROCEDURE

************************************************************************ */

void results(void){

clrscr();
cprintf("\a            THE LONGITUDINAL FIN OF TRIANGULAR PROFILE\r\n");
cprintf("    ==================================================================\r\n");
cprintf("                            Density = %4.2f kg / dm^3\r\n",density);
cprintf("                          Fin Depth = "); p_L(L, depth);
cprintf("                         Total Mass = "); p_M(M, mass);
cprintf("                         Fin Length = "); p_L(L, length);
cprintf("                         Emissivity = %4.2f\r\n", epsilon);
cprintf("                       Profile Area = "); p_A(X,Ap);
cprintf("                     Profile Number = %4.3f\r\n",PN);
cprintf("                     Base Heat Flow = "); p_H(H, qb);
cprintf("                     Fin Efficiency = %4.3f\r\n",eta);
cprintf("                    Fin Base Width  = "); p_L(d, base_width);
cprintf("                   Base Temperature = "); p_T(T,Tb);
cprintf("                   Edge Temperature = "); p_T(T,Ta);
cprintf("               Thermal Conductivity = "); p_K(K, k);
cprintf("            Environmental Parameter = %4.2f\r\n", V);
cprintf("    ==================================================================\r\n");
cprintf("                     Transmission Parameter Matrix \r\n\r\n");
cprintf("    A = %4.4f                  B = ", A); p_Z(Z,B);
cprintf("    C = "); p_Y(Y,C);
cprintf("     D = %4.4f", D);
getch();
return;}

/* ************************************************************************

      INPUT PROCEDURE

************************************************************************ */

void input(void){

clrscr(); _setcursortype(_NORMALCURSOR);
cprintf("\r\n          THE LONGITUDINAL FIN OF TRIANGULAR PROFILE\r\n");
cprintf("\r\n    ==================================================================\r\n");
cprintf("                              Fin Depth "); depth = i_L(L);
cprintf("                              Fin Length "); length = i_L(L);
cprintf("                        Fin Base Width  "); base_width = i_L(d);
cprintf("                                 Emissivity = "); scanf("%f", &epsilon);
cprintf("                      Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                   Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("         Environmental Parameter (0 to .8) = "); scanf("%f", &V);

if (S == 1) {
       cprintf("                       Output Filename is = ");
       scanf("%s", file_name);}

cprintf("\r\n    ===================================================================");

if ( epsilon > 1.0 || epsilon <= 0){
       cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
       getch(); exit(0);}

if ( V < 0 || V > 0.8){
       cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
       getch(); exit(0);}

edge_width = base_width / 1e2;
_setcursortype(_NOCURSOR);
return;}

/* ***************************************************************************

       GLOBAL FUNCTIONS

*************************************************************************** */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( hr() / (k * sin(fin_taper)));}

double u(double x){
return( 2 * sqrt(x * m() ) );}

double A_(void){
return(  ua * (I1(ua)*K0(ub) + I0(ub)*K1(ua))  );}

double B_(void){
return (  ub*ub*(I0(ub)*K0(ua)-I0(ua)*K0(ub)) / ( 2*k*b_width*depth*m()));}

double C_(void){
return ( 2*k*a_width*depth*m()*ub*(I1(ub)*K1(ua) - I1(ua)*K1(ub))/(ua) );}

double D_(void){
return( ub*( I0(ua)*K1(ub)+I1(ub)*K0(ua)) );}
```

```
/* ******************************************************************************

       PROFILE AREA Analysis Program

       Longitudinal Fin of Rectangular Profile

***************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"

#define       Yes    1
#define       No     0
#define       N      100
#define       sigma  5.66961e-8

char          file_name[20];
double        length, width, depth, starting_width, ending_width, width_inc;
double        A, B, C, D, Ta, Tb, qb=0;
float         V, K2, epsilon, k, Tbase, density, mass, Ap, error, eta;
int           T, L, M, S, H, d, K, i, X, Z, Y;
void          input(), compute();

/* **************************************************************************

       MAIN PROGRAM

*************************************************************************** */

int main(){

FILE *data_in, *out;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

width_inc = (ending_width - starting_width ) / 100;
width = starting_width;

while (width < ending_width){

       compute();

       while (qb == 0){
              width = 1.001 * width;
              compute();}

       gotoxy(19,23);
       cprintf("width = %4.1f   length = %4.1f heat = %4.1f", w_L(d,width), w_L(L,length),
              w_H(H, qb));

       if (S == 1){
              out = fopen(file_name, "a");
              fprintf(out, "%e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e\n", w_L(d,width),
                     w_T(T,Tb), w_T(T,Ta), eta, w_H(H, qb), w_M(M,mass), A, w_Z(Z,B), w_Y(Y,C), D);
              fclose(out);}

       width = width + width_inc;}

gotoxy(21,24);clreol();cprintf("\aCompleted \r\n");
clreol();
getch();
```

```
exit(0);
return 0;}

/* ****************************************************************************

       COMPUTE ROUTINE

***************************************************************************** */

void compute(void){

double         A_(), B_(), C_(), D_(), m(), hr(), Yb();
double         inc, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double         A1, B1, C1, D1, Ttip, temp, qa, Tave, segment_diff;
int            overflow, possible_overflow = No, number_of_tries=0;

length = Ap / width;

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
       Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
       Ttip = .0001 + Tave;

inc = .5 * Ttip; i = 1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

       overflow = No; i = 1; Ta = Ttip; Tb = Ta;
       qa = sigma * width * epsilon * pow(Ta, 4) * depth;

       while (i <= N && overflow == No){

              segment_diff = 10;

              while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                     A = A_(); B = B_(); C = C_(); D = D_();
                     temp = A * Ta + B * qa;
                     qb   = C * Ta + D * qa;
                     segment_diff = ( temp - Tb) / 2;
                     Tb = temp + segment_diff;

                     if (Tb > Tbase + 50 || Tb < 0){
                            overflow   = Yes; possible_overflow = Yes;
                            if (Ttip < Tbv_min ) Tbv_min = Ttip;
                            inc = (Tbv_min - Tmax) / 2;
                            Ttip = fabs(Tbv_min - inc);
                            if (Ttip < Tave) Ttip = Tave;}}

              if (overflow == No){
                     data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                     data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                     Ta = Tb; qa = qb; ++i;}}

       if (number_of_tries > 29){
              qb=0;return;}

       if (overflow == No){
              if (Tmax < Ttip)
                     if (Tb < Tbase ) Tmax = Ttip;

              overall_diff = (Tb - Tbase) / 2;

              if ( possible_overflow == Yes){
                            inc = inc / 2;
                            if (overall_diff > 0 )  Ttip = Ttip - inc;
```

```
                        else  Ttip = Ttip + inc;}
            else
                  Ttip = Ttip - overall_diff;
            number_of_tries++;
            gotoxy(27,24);
            cprintf("N = %d Ta = %4.1f Tb = %4.1f   ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
            if (Ttip < Tave) Ttip = Tave;}}

for (i = N - 1; i >= 1; i--){
      A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
      C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
      A = A1; B = B1; C=C1; D=D1;}

Ta = Ttip;
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase,4));
mass = width * length * depth * density * 1000;
return;}

/* ***************************************************************************

      INPUT PROCEDURE

*************************************************************************** */

void input(void){

clrscr(); _setcursortype(_NORMALCURSOR);
cprintf("\r\n          THE LONGITUDINAL FIN OF RECTANGULAR PROFILE");
cprintf("\r\n    ====================================================================\r\n");
cprintf("                            Fin Depth "); depth = i_L(L);
cprintf("                   Starting Fin Width "); starting_width = i_L(d);
cprintf("                     Ending Fin Width "); ending_width = i_L(d);
cprintf("                         Profile Area "); Ap = i_A(X);
cprintf("                                Emissivity = "); scanf("%f", &epsilon);
cprintf("                     Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                  Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("         Environmental Parameter (0 to .8) = "); scanf("%f", &V);

if (S == 1) {
    cprintf("                        Output Filename is = "); scanf("%s", file_name);}

cprintf("\n    ====================================================================\r\n");

if ( epsilon > 1.0 || epsilon <= 0){
      cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
      getch(); exit(0);}

if ( V < 0 || V > 0.8){
      cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
      getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* ***************************************************************************

      GLOBAL FUNCTIONS

*************************************************************************** */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( sqrt( ( 2.0 * hr()) / ( k * width) ) );}

double Yb(void){
```

```
return( sqrt( 2.0 * hr() * k * width) * depth );}

double A_(void){
return( cosh( m() * length / N )  );}

double B_(void){
return ( sinh( m() * length / N ) / Yb() );}

double C_(void){
return ( sinh( m() * length / N ) * Yb() );}

double D_(void){
return( cosh( m() * length / N )  );}
```

```
/* ****************************************************************************

      PROFILE AREA Analysis Program

      Longitudinal Fin of Trapezoidal Profile

**************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"
#include "bessel.h"

#define      Yes    1
#define      No     0
#define      N      100
#define      sigma  5.66961e-8

char         file_name[20];
double       length, edge_width, base_width, a_width, b_width, depth, starting_width, ending_width;
double       A, B, C, D, Ta, Tb, qb=0, ua, ub, fin_taper, a, width_inc;
float        V, K2, epsilon, k, Tbase, taper_ratio, density, mass,  Ap, error, eta;
int          T, L, M, S, H, d, K, i, X, Z, Y;
void         input(), compute();

/* **************************************************************************

      MAIN PROGRAM

************************************************************************** */

int main(){

FILE *data_in, *out;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

width_inc = (ending_width - starting_width ) / 100;
base_width = starting_width;

while (base_width < ending_width){

      compute();

      while (qb == 0){
            base_width = 1.001 * base_width;
            compute();}

      gotoxy(19,23);
      cprintf("width = %4.1f   length = %4.1f heat = %4.1f", w_L(d,base_width), w_L(L,length),
            w_H(H, qb));

      if (S == 1){
            out = fopen(file_name, "a");
            fprintf(out, "%e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e\n", w_L(d,base_width),
                  w_T(T,Tb), w_T(T,Ta), eta, w_H(H, qb), w_M(M,mass), A, w_Z(Z,B), w_Y(Y,C), D);
            fclose(out);}

      base_width = base_width + width_inc;}

gotoxy(21,24);clreol();cprintf("\aCompleted \r\n");
clreol();
```

```
getch();
exit(0);
return 0;}

/* ***************************************************************************

      COMPUTE ROUTINE

*************************************************************************** */

void compute(void){

double       A_(), B_(), C_(), D_(), m(), hr(), u(double x);
double       length_inc, b1, db;
double       inc, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double       A1, B1, C1, D1, Ttip, temp, qa, segment_diff;
int          overflow, possible_overflow = No, number_of_tries=0;

edge_width = base_width * taper_ratio;
length = 2 * Ap / ( base_width + edge_width );
b1 = length / ( 1 - taper_ratio );
fin_taper = atan2( base_width, 2*b1 );
db = length / N;

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      qa = sigma * edge_width * epsilon * pow(Ta, 4) * depth;
      a = b1 - length;

      while (i <= N && overflow == No){

            segment_diff = 10;
            a_width = edge_width * a / ( b1 - length);
            b_width = edge_width * ( a + db ) / ( b1 - length);

            while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                  ua = u(a) ; ub = u(a + db);
                  A = A_(); B = B_(); C = C_(); D = D_();
                  temp = A * Ta + B * qa;
                  qb   = C * Ta + D * qa;
                  segment_diff = ( temp - Tb) / 2;
                  Tb = temp + segment_diff;

                  if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

            if (overflow == No){
                  data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                  data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                  Ta = Tb; qa = qb; a = a + db; ++i;}}

      if (number_of_tries > 29){
            qb=0; return;}
```

```
       if (overflow == No){
             if (Tmax < Ttip)
                   if (Tb < Tbase) Tmax = Ttip;

             overall_diff = (Tb - Tbase) / 2;

             if ( possible_overflow == Yes){
                         inc = inc / 2;
                         if (overall_diff > 0 )  Ttip = Ttip - inc;
                         else  Ttip = Ttip + inc;}
             else
                   Ttip = Ttip - overall_diff;
             number_of_tries++;
             gotoxy(27,24);
             cprintf("N = %d Ta = %4.1f Tb = %4.1f  ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
             if (Ttip < Tave) Ttip = Tave;}}

for (i = N - 1; i >= 1; i--){
      A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
      C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
      A = A1; B = B1; C=C1; D=D1;}

Ta = Ttip;
mass = .5 * ( edge_width + base_width ) * length * depth * density * 1000;
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase, 4));
return;}

/* ***************************************************************************

      INPUT PROCEDURE

*************************************************************************** */

void input(void){

clrscr(); _setcursortype(_NORMALCURSOR);
cprintf("         THE LONGITUDINAL FIN OF TRAPEZOIDAL PROFILE");
cprintf("\r\n    ==================================================================\r\n");
cprintf("                              Fin Depth "); depth = i_L(L);
cprintf("                 Starting Base Fin Width "); starting_width = i_L(d);
cprintf("                   Ending Base Fin Width "); ending_width = i_L(d);
cprintf("                               Taper Ratio = "); scanf("%f", &taper_ratio);
cprintf("                           Profile Area "); Ap = i_A(X);
cprintf("                                  Emissivity = "); scanf("%f", &epsilon);
cprintf("                     Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                  Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("         Environmental Parameter (0 to .8) = "); scanf("%f", &V);

if (S == 1) {
    cprintf("                          Output Filename is = "); scanf("%s", file_name);}

cprintf("\n    ==================================================================\r\n");

if ( taper_ratio > .99 || taper_ratio <= 0.0) {
      cprintf("\r\n\r\nFATAL ERROR:  Taper Ratio is not between 0 and 1");
      getch(); exit(0);}

if ( starting_width >= ending_width){
      cprintf("\r\n\r\nFATAL ERROR:  Starting Base Fin Width >= Ending Base Fin Width");
      getch(); exit(0);}

if ( epsilon > 1.0 || epsilon <= 0){
      cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
      getch(); exit(0);}

if ( V < 0 || V > 0.8){
      cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
      getch(); exit(0);}
```

```
_setcursortype(_NOCURSOR);
return;}

/* *************************************************************************

      GLOBAL FUNCTIONS

************************************************************************* */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( hr() / (k * sin(fin_taper)));}

double u(double x){
return( 2 * sqrt(x * m() ) );}

double A_(void){
return(  ua * (I1(ua)*K0(ub) + I0(ub)*K1(ua))  );}

double B_(void){
return (  ub*ub*(I0(ub)*K0(ua)-I0(ua)*K0(ub)) / ( 2*k*b_width*depth*m()));}

double C_(void){
return ( 2*k*a_width*depth*m()*ub*(I1(ub)*K1(ua) - I1(ua)*K1(ub))/(ua) );}

double D_(void){
return( ub*( I0(ua)*K1(ub)+I1(ub)*K0(ua)) );}
```

```
/* ****************************************************************************

       PROFILE AREA Analysis Program

       Longitudinal Fin of Triangular Profile

**************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"
#include "bessel.h"

#define        Yes    1
#define        No     0
#define        N      99
#define        sigma  5.66961e-8

char           file_name[20];
double         length, width_edge, base_width, a_width, b_width, depth, starting_width, ending_width;
double         A, B, C, D, Ta, Tb, qb=0, ua, ub, fin_taper, a, inc_width;
float          V, K2, epsilon, k, Tbase, density, mass, Ap, error, eta;
int            T, L, M, S, H, d, K, i, X, Z, Y;
void           input(), compute();

/* ************************************************************************

       MAIN PROGRAM

************************************************************************ */

int main(){

FILE *data_in, *out;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

inc_width = (ending_width - starting_width ) / 100;
base_width = starting_width;

while (base_width < ending_width){

       compute();

       while (qb == 0){
              base_width = 1.001 * base_width;
              compute();}

       gotoxy(19,23); cprintf("width = %4.1f   length = %4.1f heat = %4.1f", w_L(d,base_width),
              w_L(L,length), w_H(H, qb));

       if (S == 1){
         out = fopen(file_name, "a");
              fprintf(out, "%e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e\n", w_L(d,base_width),
                     w_T(T,Tb), w_T(T,Ta), eta, w_H(H, qb), w_M(M,mass), A, w_Z(Z,B), w_Y(Y,C), D);
              fclose(out);}

       base_width = base_width + inc_width;}

gotoxy(21,24);clreol();cprintf("\aCompleted \r\n");
clreol();
getch();
```

```
exit(0);
return 0;}

/* ***************************************************************************

        COMPUTE ROUTINE

*************************************************************************** */

void compute(void){

double          A_(), B_(), C_(), D_(), m(), hr(), u(double x);
double          length_inc, b1, db;
double          inc, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double          A1, B1, C1, D1, Ttip, temp, qb, segment_diff;
int             overflow, possible_overflow = No, number_of_tries=0;

length = 2 * Ap / base_width;
width_edge = base_width / 100;
db = length / ( N + 1 );
b1 = length;
length = length - db;
fin_taper = atan2( base_width, 2*b1 );

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
        Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
        Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

        overflow = No; i = 1; Ta = Ttip; Tb = Ta;
        segment_diff = 10;

        while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                ub = u(db);
                temp = Ta / I0(ub);
                C    = 2 * k * width_edge * depth * pow(n(),2) * I1(ub) / (ub * I0(ub));
                qb   = C * Tb;
                segment_diff = ( temp - Tb) / 2;
                Tb = temp + segment_diff;

                if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

        if (overflow == No){
                data[i][0] = qb; data[i][1] = Tb; data[i][2] = 1;
                data[i][3] = 0;  data[i][4] = C;  data[i][5] = 1;
                a = b1 - length;
                Ta = Tb; qa = qb; ++i;}

        while (i <= N && overflow == No){

                segment_diff = 10;
                a_width = width_edge * a / ( b1 - length);
                b_width = width_edge * ( a + db ) / ( b1 - length);

                while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){
```

```
                ua = u(a) ; ub = u(a + db);
                A = A_(); B = B_(); C = C_(); D = D_();
                temp = A * Ta + B * qa;
                qb   = C * Ta + D * qa;
                segment_diff = ( temp - Tb) / 2;
                Tb = temp + segment_diff;

                if (Tb > Tbase + 50 || Tb < 0){
                      overflow   = Yes; possible_overflow = Yes;
                      if (Ttip < Tbv_min ) Tbv_min = Ttip;
                      inc = (Tbv_min - Tmax) / 2;
                      Ttip = fabs(Tbv_min - inc);
                      if (Ttip < Tave) Ttip = Tave;}}

          if (overflow == No){
                data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                Ta = Tb; qa = qb; a = a + db; ++i;}}

      if (number_of_tries > 29){
            qb=0; return;}

      if (overflow == No){
            if (Tmax < Ttip)
                  if (Tb < Tbase) Tmax = Ttip;

            overall_diff = (Tb - Tbase) / 2;

            if ( possible_overflow == Yes){
                        inc = inc / 2;
                        if (overall_diff > 0 )  Ttip = Ttip - inc;
                        else  Ttip = Ttip + inc;}
            else
                  Ttip = Ttip - overall_diff;
            number_of_tries++;
            gotoxy(27,24);
            cprintf("N = %d Ta = %4.1f Tb = %4.1f  ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
            if (Ttip < Tave) Ttip = Tave;}}

for (i = N - 1; i >= 1; i--){
      A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
      C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
      A = A1; B = B1; C=C1; D=D1;}

length = length + db;
Ta = Ttip;
mass = .5 * base_width  * length * depth * density * 1000;
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase,4));
return;}

/* ***********************************************************************

      INPUT PROCEDURE

*********************************************************************** */

void input(void){

clrscr();_setcursortype(_NORMALCURSOR);
cprintf("         THE LONGITUDINAL FIN OF TRIANGULAR PROFILE\r\n");
cprintf("    ==============================================================\r\n");
cprintf("                          Fin Depth "); depth = i_L(L);
cprintf("              Starting Base Fin Width "); starting_width = i_L(d);
cprintf("                Ending Base Fin Width "); ending_width = i_L(d);
cprintf("                      Profile Area "); Ap = i_A(X);
cprintf("                             Emissivity = "); scanf("%f", &epsilon);
cprintf("                  Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                 Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("        Environmental Parameter (0 to .8) = "); scanf("%f", &V);

if (S == 1) {
   cprintf("                       Output Filename is = "); scanf("%s", file_name);}

cprintf("\n    ===============================================================\r\n");

if ( starting_width >= ending_width){
      cprintf("\r\n\r\nFATAL ERROR:  Starting Base Fin Width >= Ending Base Fin Width");
      getch(); exit(0);}

if ( epsilon > 1.0 || epsilon <= 0){
      cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
      getch(); exit(0);}

if ( V < 0 || V > 0.8){
      cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
      getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* *************************************************************************

      GLOBAL FUNCTIONS

************************************************************************** */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( hr() / (k * sin(fin_taper)));}

double u(double x){
return( 2 * sqrt(x * m() ) );}

double A_(void){
return(  ua * (I1(ua)*K0(ub) + I0(ub)*K1(ua))  );}

double B_(void){
return (  ub*ub*(I0(ub)*K0(ua)-I0(ua)*K0(ub)) / ( 2*k*b_width*depth*m()));}

double C_(void){
return ( 2*k*a_width*depth*m()*ub*(I1(ub)*K1(ua) - I1(ua)*K1(ub))/(ua) );}

double D_(void){
return( ub*( I0(ua)*K1(ub)+I1(ub)*K0(ua)) );}
/* ***************************************************************************

      OPTIMIZE Analysis Program

      Longitudinal Fin of Rectangular Profile

**************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"

#define      Yes   1
#define      No    0
#define      N     100
#define      sigma 5.66961e-8

double       length, width, depth, A, B, C, D, Ta, Tb, qb=0;
```

```
float       Tbase, heat, PN, eta, V, K2, epsilon, k, density, mass, Ap, error;
int         T, L, M, S, H, d, K, i, X, Z, Y;
void        input(), compute(), results();

/* ************************************************************************

      MAIN PROGRAM

************************************************************************** */

int main(){

FILE *data_in;

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

heat = heat / ( 1 - V);
Ap = (1.6178 * pow(heat/depth , 3)) / ( k * pow(epsilon,2) * pow(Tbase,9) * pow(sigma,2));
width = (1.8648* pow(heat/depth , 2)) / ( k * sigma*epsilon*pow(Tbase,5));

compute();
results();
exit(0);
return 0;}

/* ************************************************************************

      COMPUTE ROUTINE

************************************************************************** */

void compute(void){

double   A_(), B_(), C_(), D_(), m(), hr(), Yb();
double   inc, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double   A1, B1, C1, D1, Ttip, Tave, temp, qa, segment_diff;
int      overflow, possible_overflow = No, number_of_tries=0;

length = Ap / width;
K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = .0001 + Tave;

inc = .5 * Ttip; i = 1;

while (fabs(overall_diff) > ( Tbase * error ) || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      qa = sigma * width * epsilon * pow(Ta, 4) * depth;

      while (i <= N && overflow == No){

            segment_diff = 10;

      while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

            A = A_(); B = B_(); C = C_(); D = D_();
            temp = A * Ta + B * qa;
            qb   = C * Ta + D * qa;
            segment_diff = ( temp - Tb) / 2;
            Tb = temp + segment_diff;

            if (Tb > Tbase + 50 || Tb < 0){
                  overflow   = Yes; possible_overflow = Yes;
                  if (Ttip < Tbv_min ) Tbv_min = Ttip;
                  inc = (Tbv_min - Tmax) / 2 ;
                  Ttip = fabs(Tbv_min - inc);
                  if (Ttip < Tave) Ttip = Tave;}}

            if (overflow == No){
                  data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                  data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                  Ta = Tb; qa = qb; ++i;}}

      if (number_of_tries > 29){
            clrscr(); cprintf("Fatal Error:  Unable to Compute");
            getch(); exit(0);}

      if (overflow == No){
            if (Tmax < Ttip)
                  if (Tb < Tbase) Tmax = Ttip;

            overall_diff = (Tb - Tbase) / 2;

            if ( possible_overflow == Yes){
                  inc = inc / 2;
                  if (overall_diff > 0 )  Ttip = Ttip - inc;
                        else  Ttip = Ttip + inc;}
            else
                  Ttip = Ttip - overall_diff;
            number_of_tries++;
            gotoxy(27,24);
            cprintf("N = %d Ta = %4.1f Tb = %4.1f     ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
            if (Ttip < Tave) Ttip = Tave;}}

for (i = N - 1; i >= 1; i--){
      A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
      C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
      A = A1; B = B1; C=C1; D=D1;}

Ta = Ttip;
eta = qb / (2 * sigma * epsilon * length * depth * pow(Tbase,4));
PN = length * sqrt(sigma * epsilon * pow(Tbase,3) / (k * width));
mass = width * length * depth * density * 1000;
return;}

/* ***************************************************************************

      OUTPUT PROCEDURE

*************************************************************************** */

void results(void){

clrscr();
cprintf("\a                         OPTIMUM PARAMETERS");
cprintf("\r\n         THE LONGITUDINAL FIN OF RECTANGULAR PROFILE");
cprintf("\r\n    ================================================================\r\n");
cprintf("                         Density = %4.2f kg / dm^3\r\n",density);
cprintf("                       Fin Depth = "); p_L(L, depth);
cprintf("                      Total Mass = "); p_M(M, mass);
cprintf("                      Fin Height = "); p_L(L, length);
cprintf("                      Fin Width  = "); p_L(d, width);
cprintf("                      Emissivity = %4.2f\r\n", epsilon);
cprintf("                    Profile Area = "); p_A(X, Ap);
cprintf("                  Profile Number = %4.3f\r\n",PN);
cprintf("                  Base Heat Flow = "); p_H(H, qb);
cprintf("                  Fin Efficiency = %4.3f\r\n",eta);
cprintf("                Base Temperature = "); p_T(T,Tb);
cprintf("               Edge Temperature = "); p_T(T,Ta);
cprintf("           Thermal Conductivity = "); p_K(K, k);
cprintf("        Environmental Parameter = %4.2f\r\n", V);
cprintf("    ====================================================================\r\n");
cprintf("                 Transmission Parameter Matrix \r\n\r\n");
cprintf("    A = %4.4f                    B = ", A); p_Z(Z,B);
cprintf("    C = "); p_Y(Y,C);
cprintf("     D = %4.4f", D);
getch();
return;}

/* *************************************************************************

      INPUT PROCEDURE

************************************************************************* */

void input(void){

clrscr(); _setcursortype(_NORMALCURSOR);
cprintf("\r\n          THE LONGITUDINAL FIN OF RECTANGULAR PROFILE\r\n");
cprintf("\r\n    ====================================================================\r\n");
cprintf("                             Fin Depth "); depth = i_L(L);
cprintf("                   Heat Dissipation "); heat = i_H(H);
cprintf("                              Emissivity = "); scanf("%f", &epsilon);
cprintf("                      Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                   Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("         Environmental Parameter (0 to .8) = "); scanf("%f", &V);
cprintf("\r\n    ====================================================================");

if ( epsilon > 1.0 || epsilon <= 0){
      cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
      getch(); exit(0);}

if ( V < 0 || V > 0.8){

      cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
      getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* *************************************************************************

      GLOBAL FUNCTIONS

************************************************************************* */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( sqrt( ( 2.0 * hr()) / ( k * width) ) );}

double Yb(void){
return( sqrt( 2.0 * hr() * k * width) * depth );}

double A_(void){
return( cosh( m() * length / N )  );}

double B_(void){
return ( sinh( m() * length / N ) / Yb() );}

double C_(void){
return ( sinh( m() * length / N ) * Yb() );}

double D_(void){
```

```
return( cosh( m() * length / N )  );}
```

```
/* ****************************************************************************

      OPTIMIZE Analysis Program

      Longitudinal Fin of Trapezoidal Profile

***************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"
#include "bessel.h"

#define       Yes    1
#define       No     0
#define       N      100
#define       sigma  5.66961e-8

char          file_name[20];
double        length, edge_width, a_width, b_width, base_width, depth, inc_width;
double        A, B, C, D, Ta, a, Tb, qb=0, ua, ub, fin_taper, Constant_3;
float         K2, taper_ratio, V, eta, PN, epsilon, heat, k, Tbase, density, mass, Ap, error;
int           T, L, M, S, H, d, K, i, X, Z, Y;
void          input(), compute(), results();
FILE          *data_in, *out;

/* ************************************************************************

      MAIN PROGRAM

************************************************************************* */

int main(){

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();

heat = heat / (1 - V);
Constant_3 = 0.3881*pow(taper_ratio,2) - 1.0324*taper_ratio + 2.7522;
Ap       =  ((0.5147 * taper_ratio + 1.1382) * pow(heat/depth , 3)) / ( k * pow(epsilon,2) *
                     pow(Tbase,9) * pow(sigma,2));
base_width = (Constant_3 * pow(heat/depth , 2)) / ( k * sigma*epsilon*pow(Tbase,5));
edge_width = taper_ratio * base_width;

compute();
results();
exit(0);
return 0;}

/* ************************************************************************

      COMPUTE ROUTINE

************************************************************************* */

void compute(void){

double        A_(), B_(), C_(), D_(), m(), hr(), u(double x);
double        length_inc, b1, db;
double        inc, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000, data[N+1][6];
double        A1, B1, C1, D1, Ttip, temp, qa, segment_diff;
int           overflow, possible_overflow = No, number_of_tries=0;
```

```
length = 2 * Ap / ( base_width + edge_width );
b1 = length / ( 1 - taper_ratio );
fin_taper = atan2( base_width, 2*b1 );
db = length / N;

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      qa = sigma * edge_width * epsilon * pow(Ta, 4) * depth;
      a = b1 - length;

      while (i <= N && overflow == No){

            segment_diff = 10;
            a_width = edge_width * a / ( b1 - length);
            b_width = edge_width * ( a + db ) / ( b1 - length);

            while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                  ua = u(a) ; ub = u(a + db);
                  A = A_(); B = B_(); C = C_(); D = D_();
                  temp = A * Ta + B * qa;
                  qb   = C * Ta + D * qa;
                  segment_diff = ( temp - Tb) / 2;
                  Tb = temp + segment_diff;

                  if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

            if (overflow == No){
                  data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                  data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                  Ta = Tb; qa = qb; a = a + db; ++i;}}

      if (number_of_tries > 29){
            clrscr(); cprintf("Fatal Error:  Unable to Compute");
            getch(); exit(0);}

      if (overflow == No){
            if (Tmax < Ttip)
                  if (Tb < Tbase) Tmax = Ttip;

            overall_diff = (Tb - Tbase) / 2;

            if ( possible_overflow == Yes){
                        inc = inc / 2;
                        if (overall_diff > 0 )  Ttip = Ttip - inc;
                        else  Ttip = Ttip + inc;}
            else
                  Ttip = Ttip - overall_diff;
            number_of_tries++;
            gotoxy(27,24);
            cprintf("N = %d Ta = %4.1f Tb = %4.1f  ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
            if (Ttip < Tave) Ttip = Tave;}}
```

```
for (i = N - 1; i >= 1; i--){
      A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
      C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
      A = A1; B = B1; C=C1; D=D1;}

if (S == 1){
      out = fopen(file_name, "w");
      for (i = 1; i <= N; i++)
            fprintf(out, "%e,  %e,  %e\n", w_L(d, i * length / N), w_T(T,data[i][1]), w_H(H,
                  data[i][0]) );
      fclose(out);}

Ta = Ttip;
Ap = .5 * length * ( edge_width + base_width );
mass = Ap * depth * density * 1000;
PN = 2 * sigma * epsilon * pow(length, 2) * pow(Tbase,3) / (k* base_width);
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase,4));
return;}

/* **********************************************************************

      OUTPUT PROCEDURE

********************************************************************** */

void results(void){

clrscr();
cprintf("\a\r\n                         OPTIMUM PARAMETERS");
cprintf("\r\n         THE LONGITUDINAL FIN OF TRAPEZOIDAL PROFILE");
cprintf("\r\n    ==================================================================\r\n");
cprintf("                         Density = %4.2f kg / dm^3\r\n",density);
cprintf("                       Fin Depth = "); p_L(L, depth);
cprintf("                      Total Mass = "); p_M(M, mass);
cprintf("                      Fin Length = "); p_L(L, length);
cprintf("                      Emissivity = %4.2f\r\n", epsilon);
cprintf("                    Profile Area = "); p_A(X,Ap);
cprintf("                  Profile Number = %4.3f\r\n",PN);
cprintf("                  Base Heat Flow = "); p_H(H, qb);
cprintf("                  Fin Efficiency = %4.3f\r\n",eta);
cprintf("                 Fin Edge Width  = "); p_L(d, edge_width);
cprintf("                 Fin Base Width  = "); p_L(d, base_width);
cprintf("                Base Temperature = "); p_T(T,Tb);
cprintf("                Edge Temperature = "); p_T(T,Ta);
cprintf("            Thermal Conductivity = "); p_K(K, k);
cprintf("         Environmental Parameter = %4.2f\r\n", V);
cprintf("    ==================================================================\r\n");
cprintf("                  Transmission Parameter Matrix \r\n\r\n");
cprintf("    A = %4.4f                  B = ", A); p_Z(Z,B);
cprintf("    C = "); p_Y(Y,C);
cprintf("     D = %4.4f", D);
getch();
return;}

/* **********************************************************************

      INPUT PROCEDURE

********************************************************************** */

void input(void){

clrscr(); _setcursortype(_NORMALCURSOR);
cprintf("\r\n        THE LONGITUDINAL FIN OF TRAPEZOIDAL PROFILE\r\n");
cprintf("\r\n   ================================================================\r\n");
cprintf("                           Fin Depth "); depth = i_L(L);
cprintf("                      Heat Dissipation "); heat = i_H(H);
cprintf("                           Taper Ratio = "); scanf("%f",&taper_ratio);
cprintf("                            Emissivity = "); scanf("%f", &epsilon);
cprintf("                    Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                 Base Temperature "); Tbase = i_T(T);
cprintf("   Thermal Conductivity "); k = i_K(K);
cprintf("        Environmental Parameter (0 to .8) = "); scanf("%f", &V);
cprintf("\n=================================================================\r\n");

if ( epsilon > 1.0 || epsilon <= 0){
       cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
       getch(); exit(0);}

if ( V < 0 || V > 0.8){
       cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
       getch(); exit(0);}

if ( taper_ratio > .99 || taper_ratio <= 0){
    cprintf("\r\n\r\nFATAL ERROR:  Taper Ratio is not between 0 and 1.0");
       getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* ************************************************************************

       GLOBAL FUNCTIONS

************************************************************************** */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( hr() / (k * sin(fin_taper)));}

double u(double x){
return( 2 * sqrt(x * m() ) );}

double A_(void){
return(  ua * (I1(ua)*K0(ub) + I0(ub)*K1(ua))  );}

double B_(void){
return (  ub*ub*(I0(ub)*K0(ua)-I0(ua)*K0(ub)) / ( 2*k*b_width*depth*m()));}

double C_(void){
return ( 2*k*a_width*depth*m()*ub*(I1(ub)*K1(ua) - I1(ua)*K1(ub))/(ua) );}

double D_(void){
return( ub*( I0(ua)*K1(ub)+I1(ub)*K0(ua)) );}
```

```
/* ****************************************************************************

      OPTIMIZE Analysis Program

      Longitudinal Fin of Triangular Profile

*************************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "units.h"
#include "bessel.h"

#define       Yes    1
#define       No     0
#define       N      99
#define       sigma  5.66961e-8

char          file_name[20];
double        length, edge_width, base_width, a_width, b_width, depth;
double        A, B, C, D, Ta, Tb, qb=0, ua, ub, fin_taper, a;
float         V, K2, epsilon, k, Tbase, density, mass, Ap, error, eta, PN, heat;
int           T, L, M, S, H, d, K, i, X, Z, Y;
void          input(), results(), compute();
FILE          *data_in, *out;

/* *************************************************************************

      MAIN PROGRAM

************************************************************************* */

int main(){

data_in = fopen("units.ext","r");
fscanf(data_in,"%d %d %d %d %d %d %d %d %d %d %f",&T,&L,&M,&S,&H,&d,&K,&X,&Z,&Y,&error);
fclose(data_in);
error = error / 100;

input();
heat = heat / (1 - V);
Ap      = (1.1234 * pow(heat/depth , 3)) / ( k * pow(sigma*epsilon,2) * pow(Tbase,9));
base_width = (2.4852 * pow(heat/depth , 2)) / ( k * sigma*epsilon*pow(Tbase,5));

compute();
results();
exit(0);
return 0;}

/* *************************************************************************

    COMPUTE ROUTINE

************************************************************************* */

void compute(){

double        A_(), B_(), C_(), D_(), m(), hr(), u(double x);
double        length_inc, b1, db;
double        inc, Tave, overall_diff = 100, Tmax=0, Tbv_min = 100000, data[N+1][6];
double        A1, B1, C1, D1, Ttip, temp, qa, segment_diff;
int           overflow, possible_overflow = No, number_of_tries = 0;


length = 2 * Ap / base_width;
edge_width = base_width / 100;
```

```
db = length / ( N + 1 );
b1 = length;
length = length - db;
fin_taper = atan2( base_width, 2*b1 );

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      segment_diff = 10;

      while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

            ub = u(db);
            temp = Ta / I0(ub);
            C    = 2 * k * edge_width * depth * pow(m(),2) * I1(ub) / (ub * I0(ub));
            qb   = C * temp;
            segment_diff = ( temp - Tb) / 2;
            Tb = temp + segment_diff;

            if (Tb > Tbase + 50 || Tb < 0){
                  overflow   = Yes; possible_overflow = Yes;
                  if (Ttip < Tbv_min ) Tbv_min = Ttip;
                  inc = (Tbv_min - Tmax) / 2;
                  Ttip = fabs(Tbv_min - inc);
                  if (Ttip < Tave) Ttip = Tave;}}

      if (overflow == No){
            data[i][0] = qb; data[i][1] = Tb; data[i][2] = 1;
            data[i][3] = 0;  data[i][4] = C;  data[i][5] = 1;
            Ta = Tb; qa = qb; ++i;}

      a = b1 - length;

      while (i <= N && overflow == No){

            segment_diff = 10;
            a_width = edge_width * a / ( b1 - length);
            b_width = edge_width * ( a + db ) / ( b1 - length);

            while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                  ua = u(a) ; ub = u(a + db);
                  A = A_(); B = B_(); C = C_(); D = D_();
                  temp = A * Ta + B * qa;
                  qb   = C * Ta + D * qa;
                  segment_diff = ( temp - Tb) / 2;
                  Tb = temp + segment_diff;

                  if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

            if (overflow == No){
                  data[i][0] = qb; data[i][1] = Tb; data[i][2] = A;
                  data[i][3] = B;  data[i][4] = C;  data[i][5] = D;
                  Ta = Tb; qa = qb; a = a + db; ++i;}}
```

```
        if (number_of_tries > 29){
              clrscr(); cprintf("Fatal Error:  Unable to Compute");
              getch(); exit(0);}

        if (overflow == No){
              if (Tmax < Ttip)
                    if (Tb < Tbase) Tmax = Ttip;

              overall_diff = (Tb - Tbase) / 2;

              if ( possible_overflow == Yes){
                          inc = inc / 2;
                          if (overall_diff > 0 )  Ttip = Ttip - inc;
                          else  Ttip = Ttip + inc;}
              else
                    Ttip = Ttip - overall_diff;
              number_of_tries++;
              gotoxy(27,24);
              cprintf("N = %d Ta = %4.1f Tb = %4.1f  ",number_of_tries,w_T(T,Ttip),w_T(T,Tb));
              if (Ttip < Tave) Ttip = Tave;}}

for (i = N - 1; i >= 1; i--){
        A1 = A * data[i][2] + B * data[i][4]; B1 = A * data[i][3] + B * data[i][5];
        C1 = C * data[i][2] + D * data[i][4]; D1 = C * data[i][3] + D * data[i][5];
        A = A1; B = B1; C=C1; D=D1;}

length = length + db;

if (S == 1){
        out = fopen(file_name, "w");
        for (i = 1; i <= N; i++)
              fprintf(out, "%e,  %e,  %e\n", w_L(d, i * length / N), w_T(T,data[i][1]), w_H(H,
                    data[i][0]));
    fclose(out);}

Ta = Ttip;
mass = Ap * depth * density * 1000;
PN = 2 * sigma * epsilon * pow(length,2) * pow(Tbase,3) / (k*base_width);
eta = qb / ( 2 * sigma * epsilon * length * depth * pow(Tbase,4));
return;}

/* *********************************************************************

        OUTPUT PROCEDURE

********************************************************************* */

void results(void){

clrscr();
cprintf("\a                          OPTIMUM PARAMETERS");
cprintf("\r\n          THE LONGITUDINAL FIN OF TRIANGULAR PROFILE");
cprintf("\r\n    ============================================================\r\n");
cprintf("                        Density = %4.2f kg / dm^3\r\n",density);
cprintf("                      Fin Depth = "); p_L(L, depth);
cprintf("                     Total Mass = "); p_M(M, mass);
cprintf("                     Fin Length = "); p_L(L, length);
cprintf("                     Emissivity = %4.2f\r\n", epsilon);
cprintf("                   Profile Area = "); p_A(X,Ap);
cprintf("                 Profile Number = %4.3f\r\n",PN);
cprintf("                 Base Heat Flow = "); p_H(H, qb);
cprintf("                 Fin Efficiency = %4.3f\r\n",eta);
cprintf("                Fin Base Width  = "); p_L(d, base_width);
cprintf("               Base Temperature = "); p_T(T,Tb);
cprintf("               Edge Temperature = "); p_T(T,Ta);
cprintf("           Thermal Conductivity = "); p_K(K, k);
cprintf("        Environmental Parameter = %4.2f\r\n", V);
cprintf("    ============================================================\r\n");
cprintf("                    Transmission Parameter Matrix \r\n\r\n");
cprintf("    A = %4.4f                       B = ", A); p_Z(Z,B);
cprintf("    C = "); p_Y(Y,C);
cprintf("     D = %4.4f", D);
getch();
return;}

/* ***************************************************************************

       INPUT PROCEDURE

*************************************************************************** */

void input(void){

clrscr();_setcursortype(_NORMALCURSOR);
cprintf("\r\n          THE LONGITUDINAL FIN OF TRIANGULAR PROFILE");
cprintf("\r\n    ===================================================================\r\n");
cprintf("                          Fin Depth "); depth = i_L(L);
cprintf("                    Heat Dissipation "); heat = i_H(H);
cprintf("                               Emissivity = "); scanf("%f", &epsilon);
cprintf("                    Density ( kg / dm^3 ) = "); scanf("%f", &density);
cprintf("                 Base Temperature "); Tbase = i_T(T);
cprintf("    Thermal Conductivity "); k = i_K(K);
cprintf("         Environmental Parameter (0 to .8) = "); scanf("%f", &V);
cprintf("\n=====================================================================\r\n");

if ( epsilon > 1.0 || epsilon <= 0){
       cprintf("\r\n\r\nFATAL ERROR:  Emissivity is not between 0 and 1");
       getch(); exit(0);}

if ( V < 0 || V > 0.8){
       cprintf("\r\n\r\nFATAL ERROR:  Environmental Parameter is not between 0 and 0.8");
       getch(); exit(0);}

_setcursortype(_NOCURSOR);
return;}

/* ***************************************************************************

       GLOBAL FUNCTIONS

*************************************************************************** */

double hr(void){
float Tav = (Tb + Ta) / 2;
return(sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav)));}

double m(void){
return( hr() / (k * sin(fin_taper)));}

double u(double x){
return( 2 * sqrt(x * m() ) );}

double A_(void){
return(  ua * (I1(ua)*K0(ub) + I0(ub)*K1(ua))  );}

double B_(void){
return (  ub*ub*(I0(ub)*K0(ua)-I0(ua)*K0(ub)) / ( 2*k*b_width*depth*m()));}

double C_(void){
return ( 2*k*a_width*depth*m()*ub*(I1(ub)*K1(ua) - I1(ua)*K1(ub))/(ua) );}

double D_(void){
return( ub*( I0(ua)*K1(ub)+I1(ub)*K0(ua)) );}
```

```
/* ****************************************************************************

      UNIT CONVERSION Program Header File

****************************************************************************** */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include "units.h"

float i_T(int T){
float V;
switch (T){
      case 0:       cprintf("(deg K) = "); scanf("%f", &V); break;
      case 1:       cprintf("(deg C) = "); scanf("%f", &V); V = V + 273.15; break;
      case 2:       cprintf("(deg F) = ");    scanf("%f", &V); V = 5 * ( V + 459.4 ) / 9; break;
      case 3:       cprintf("(deg R) = "); scanf("%f", &V); V = 5 * V / 9;}
return(V);}

void p_T(int T, float V){
switch (T){
      case 0:       cprintf("%4.2f deg K\r\n", V); break;
      case 1:       V = V - 273.15; cprintf("%4.2f deg C\r\n", V); break;
      case 2:       V = 9 * ( V - 255.222222) / 5; cprintf("%4.2f deg F\r\n", V); break;
      case 3:       V = 9 * V / 5; cprintf("%4.2f deg R\r\n", V);}
return;}

float w_T(int T, float V){
switch (T){
      case 0:       break;
      case 1:       V = V - 273.15; break;
      case 2:       V = 9 * ( V - 255.222222) / 5; break;
      case 3:       V = 9 * V / 5;}
return(V);}

void p_Z(int Z, float V){
switch (Z){
      case 0:       cprintf("%4.4f deg C   / watt\r\n", V); break;
      case 1:       V = 0.1760831 * V ; cprintf("%4.34 deg F hr / BTU\r\n", V);}
return;}

float w_Z(int Z, float V){
switch (Z){
      case 0:       break;
      case 1:       V = 0.1760831 * V;}
return(V);}

void p_Y(int Y, float V){
switch (Y){
      case 0:       cprintf("%4.4f watt   / deg C", V); break;
      case 1:       V = V / 0.1760831 ; cprintf("%4.4f BTU / hr deg F", V);}
return;}

float w_Y(int Y, float V){
switch (Y){
      case 0:       break;
      case 1:       V = V / 0.1760831;}
return(V);}

float i_L(int L){
float V;
switch (L){
      case 0:       cprintf(" (m) = "); scanf("%f", &V); break;
      case 1:       cprintf("(cm) = "); scanf("%f", &V); V = V / 100; break;
      case 2:       cprintf("(ft) = "); scanf("%f", &V); V = 0.3048 * V; break;
      case 3:       cprintf("(in) = "); scanf("%f", &V); V = 0.0254 * V; break;
      case 4:       cprintf("(mm) = "); scanf("%f", &V); V = V / 1000;}
```

```
return(V);}

void p_L(int L, float V){
switch (L){
      case 0:      cprintf("%4.2f m\r\n", V); break;
      case 1:      V = 100 * V; cprintf("%4.2f cm\r\n", V); break;
      case 2:      V = V / 0.3048; cprintf("%4.2f ft\r\n", V); break;
      case 3:      V = V / 0.0254; cprintf("%4.2f in\r\n", V); break;
      case 4:      V = 1000 * V; cprintf("%4.2f mm\r\n", V);}
return;}

float w_L(int L, float V){
switch (L){
      case 0:      break;
      case 1:      V = 100 * V; break;
      case 2:      V = V / 0.3048; break;
      case 3:      V = V / 0.0254; break;
      case 4:      V = 1000 * V;}
return(V);}

float i_A(int A){
float V;
switch (A){
      case 0:      cprintf(" (m^2) = "); scanf("%f", &V); break;
      case 1:      cprintf("(cm^2) = "); scanf("%f", &V); V = V / 10000; break;
      case 2:      cprintf("(ft^2) = "); scanf("%f", &V); V = V / 10.76; break;
      case 3:      cprintf("(in^2) = "); scanf("%f", &V); V = V / 1550; break;
      case 4:      cprintf("(mm^2) = "); scanf("%f", &V); V = V / 1000000;}
return(V);}

void p_A(int A, float V){
switch (A){
      case 0:      cprintf("%4.2 m ^ 2\r\n", V); break;
      case 1:      V = 10000 * V; cprintf("%4.2f cm ^ 2\r\n", V); break;
      case 2:      V = V * 10.76; cprintf("%4.2f ft ^ 2\r\n", V); break;
      case 3:      V = V * 1550; cprintf("%4.2f in ^ 2\r\n", V); break;
      case 4:      V = 1000000 * V; cprintf("%4.2f mm ^ 2\r\n", V);}
return;}

float w_A(int A, float V){
switch (A){
      case 0:      break;
      case 1:      V = 10000 * V; break;
      case 2:      V = V * 10.76; break;
      case 3:      V = V * 1550; break;
      case 4:      V = 1000000 * V;}
return(V);}

float i_M(int M){
float V;
switch (M){
      case 0:      cprintf(" (kg) = "); scanf("%f", &V); break;
      case 1:      cprintf(" (gm) = "); scanf("%f", &V); V = V / 1000; break;
      case 2:      cprintf("(lbm) = "); scanf("%f", &V); V = V / 2.204622; break;
      case 3:      cprintf(" (oz) = "); scanf("%f", &V); V = V / 35.27396; break;}
return(V);}

void p_M(int M, float V){
switch (M){
      case 0:      cprintf("%4.2f kg\r\n", V); break;
      case 1:      V = 1000 * V; cprintf("%4.2f grams\r\n", V); break;
      case 2:      V = 2.204622 * V; cprintf("%4.2f lbm\r\n", V); break;
      case 3:      V = 35.27396 * V; cprintf("%4.2f oz\r\n", V);}
return;}

float w_M(int M, float V){
switch (M){
      case 0:      break;
      case 1:      V = 1000 * V; break;
      case 2:      V = 2.204622 * V; break;
      case 3:      V = 35.27396 * V;}
return(V);}

float i_H(int H){
float V;
switch (H){
      case 0:      cprintf("(watts) = "); scanf("%f", &V); break;
      case 1:      cprintf("     (kw) = "); scanf("%f", &V); V = 1000 * V; break;
      case 2:      cprintf("(BTU/sec) = "); scanf("%f", &V); V = V * 1054.3502; break;
      case 3:      cprintf(" (BTU/hr) = "); scanf("%f", &V); V = V / 3.414425;}
return(V);}

void p_H(int H, float V){
switch (H){
      case 0:      cprintf("%4.2f watts\r\n", V); break;
      case 1:      V = V / 1000; cprintf("%4.2f kw\r\n", V); break;
      case 2:      V = V / 1054.3502; cprintf("%4.2f BTU / sec\r\n", V);break;
      case 3:      V = 3.414425 * V; cprintf("%4.2f BTU / hr\r\n", V);}
return;}

float w_H(int H, float V){
switch (H){
      case 0:      break;
      case 1:      V = V / 1000; break;
      case 2:      V = V / 1054.3502; break;
      case 3:      V = 3.414425 * V;}
return(V);}

float i_K(int K){
float V;
switch (K){
      case 0:      cprintf("(watts / m deg K) = "); scanf("%f", &V); break;
      case 1:      cprintf("(BTU / ft hr deg F) = "); scanf("%f", &V); V = V / 1.731;}
return(V);}

void p_K(int K, float V){
switch (K){
      case 0:      cprintf("%4.2f watts / m deg K\r\n", V); break;
      case 1:      V = 1.731 * V; cprintf("%4.2f BTU / ft hr deg F\r\n", V);}
return;}
```

```
/* ****************************************************************************

      BESSEL FUNCTION Program Header File[16]

/* ****************************************************************************

#include <stdio.h>
#include <math.h>
#include "bessel.h"

double K0(float x){

double        y, z;
double        P1=-0.57721566, P2=0.4227842, P3=0.23069756, P4=0.0348859, P5=0.00262698,
              P6=0.0001075, P7=0.0000074;
double        Q1=1.25331414, Q2=-0.07832358, Q3=0.02189568, Q4=-0.01062446, Q5=0.00587872,
              Q6=-0.00251546;
double        Q7=0.00053208;

if ( x < 2 ){
      y = ( x * x ) / 4;
      z = (-log(x/2)*I0(x))+(P1+y*(P2+y*(P3+y*(P4+y*(P5+y*(P6+y*P7))))));}
else{
      y = 2 / x;
      z = (exp(-x)/sqrt(x))*(Q1+y*(Q2+y*(Q3+y*(Q4+y*(Q5+y*(Q6+y*Q7))))));}
return(z);}

double K1(float x){

double        y, z;
double        P1=1.0, P2=0.15443144, P3=-.67278579, P4=-0.18156897, P5=-0.01919402, P6=-0.00110404,
              P7=-0.00004686;
double        Q1=1.25331414, Q2=0.23498619, Q3=-0.03655620, Q4=0.01504268, Q5=-0.00780353,
              Q6=0.00325614;
double        Q7=-0.00068245;

if ( x < 2 ){
      y = ( x * x ) / 4;
      z = (log(x/2)*I1(x))+(1/x)*(P1+y*(P2+y*(P3+y*(P4+y*(P5+y*(P6+y*P7))))));}
else{
      y = 2 / x;
      z = (exp(-x)/sqrt(x))*(Q1+y*(Q2+y*(Q3+y*(Q4+y*(Q5+y*(Q6+y*Q7))))));}
return(z);}


double I0(float x){

double        y, z, ax;
double        P1=1, P2=3.5156229, P3=3.0899424, P4=1.2067492, P5=0.2659732, P6=0.0360768,
              P7=0.0045813;
double        Q1=0.39894228, Q2=0.01328592, Q3=0.00225319, Q4=-0.00157565, Q5=0.00916281,
              Q6=-0.02057706;
double        Q7=0.02635537, Q8=-0.01647633, Q9=0.00392377;

if (fabs(x) < 3.75 ){
      y = ( x / 3.75 ) * ( x / 3.75 );
      z = P1+y*(P2+y*(P3+y*(P4+y*(P5+y*(P6+y*P7)))));}
else{
      ax = fabs(x);
      y = 3.75 / ax;
      z = (exp(ax)/sqrt(ax))*(Q1+y*(Q2+y*(Q3+y*(Q4+y*(Q5+y*(Q6+y*(Q7+y*(Q8+y*Q9))))))));}
return(z);}

double I1(float x){

double        y, z, ax;
double        P1=0.5, P2=0.87890594, P3=0.51498869, P4=0.15084934, P5=0.02658733, P6=0.00301532,
              P7=0.00032411;
double        Q1=0.39894228, Q2=-0.03988024, Q3=-0.00362018, Q4=0.00163801, Q5=-0.01031555,
                Q6=0.02282967;
double          Q7=-0.02895312, Q8=0.01787654, Q9=-0.00420059;

if (fabs(x) < 3.75 ){
        y = ( x / 3.75 ) * ( x / 3.75 );
        z = x*(P1+y*(P2+y*(P3+y*(P4+y*(P5+y*(P6+y*P7))))));}
else{
        ax = fabs(x);
        y = 3.75 / ax;
        z = (exp(ax)/sqrt(ax))*(Q1+y*(Q2+y*(Q3+y*(Q4+y*(Q5+y*(Q6+y*(Q7+y*(Q8+y*Q9))))))));}
return(z);}
```

APPENDIX B COMPUTER CODE TO DERIVE OPTIMUM GEOMETRY

```
/* ****************************************************************************

      PROFILE AREA Analysis Program To Determine The Optimum Geometry

      Longitudinal Fin of Rectangular Profile

**************************************************************************** */

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>

#define      Yes    1
#define      No     0
#define      N      100
#define      sigma  5.66961e-8

double       length, width_base, max_length;
double       A, B, C, D, Ta, Tb, qb=0, Heat, qb_max, db, Continue;
float        V, K2, epsilon, k, Tbase, Ap, taper_ratio, depth, error=0.01;
int          i, j, number_of_passes, failures;
void         compute();

/* ************************************************************************

      MAIN PROGRAM

************************************************************************ */

int main(){
float dq;
FILE *out;
randomize();
error = error / 100;
clrscr();

for (j=1; j<102; j++){

      number_of_passes = 0;
      epsilon = (float) ( random(98) );
      epsilon = .01 + epsilon / 100;
      depth =  (float) (20 + random(200));
      depth = depth / 100;
      k = 10 + random(200);
      Tbase = 100 + random(500);
      Heat = (float) (25 + random(1000));
      Heat = Heat / depth;
      V = (float) ( random(80));
      V = .01 + V / 100;
      V = 0;
      taper_ratio = (float) (random(98));
      taper_ratio = 0.01 + taper_ratio / 100;
      taper_ratio = 1;

      Continue = Yes;
      qb_max = -1;
      qb = 0;
```

```
        dq = 0;
        Ap = 1.635 * pow(Heat,3) / ( k * pow(sigma,2) * pow(epsilon,2) * pow(Tbase,9));
        length = .87 * Heat / ( sigma * epsilon * pow(Tbase,4));
        db = length / 20016;
        length = length - db;
        width_base = Ap / length;

        if (depth < 10*width_base) Continue = No;
        if (length < 10*width_base) Continue = No;

        while (Continue == Yes){

                length = length + db;
                width_base = Ap / length;
                number_of_passes++;
                failures = 0;

                compute();

                while (qb == 0 && Continue == Yes){
                        failures++;
                        length = length + db / 2;
                        width_base =  Ap / length;
                        compute();
                        if (failures == 3){
                                Continue = No;
                                number_of_passes = 1;}}

                dq = qb - dq;
                cprintf("\r\nwidth_base = %4.3f length = %4.6f heat = %4.6f dq = %4.6f",
                        width_base*1000, length, qb, dq);
                dq = qb;

                if (qb_max < qb ){
                        max_length = length;
                        qb_max = qb;}
                else
                        Continue = No;}

        if (number_of_passes > 2){
                length = max_length;
                width_base = Ap / length;
                compute();
                cprintf("\r\nWriting Rec # %d d = %4.3f  b = %4.6f qb = %4.3f Tb = %4.1f Ap = %4.1f",
                        j, width_base, length, qb, Tbase, Ap);
                out = fopen("OPT_RECT.MAT","a");
                fprintf(out, "%d,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e\r\n", j,
                   depth, length, Ap, k, Tbase, epsilon, V, qb, taper_ratio, width_base, width_base );
                fclose(out);}
        else{
                cprintf("\r\nIgnoring This Set Of Parameters");
                j=j-1;}}

return 0;}

/* ***********************************************************************

        COMPUTE ROUTINE

************************************************************************ */

void compute(void){

double          Tav, inc, overall_diff = 100, Tmax = 0, Tbv_min = 100000;
double          A, B, C, D, m, hr, Yo, Ttip, temp, qa, Tave, segment_diff;
int             overflow, possible_overflow = No, number_of_tries=0;

width_base = Ap / length;
```

```
K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
      Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
      Ttip = .0001 + Tave;

inc = .5 * Ttip; i = 1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      qa = sigma * width_base * epsilon * pow(Ta, 4) * depth;

      while (i <= N && overflow == No){

            segment_diff = 10;

            while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                        Tav = (Tb + Ta) / 2;
                        hr = sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav));
                        m = sqrt( ( 2.0 * hr) / ( k * width_base) );
                        Yo = sqrt( 2.0 * hr * k * width_base) * depth;
                        A = cosh( m * length / N );
                        B = sinh( m * length / N ) / Yo;
                        C = sinh( m * length / N ) * Yo;
                        D = A;

                  temp = A * Ta + B * qa;
                  qb   = C * Ta + D * qa;
                  segment_diff = ( temp - Tb) / 2;
                  Tb = temp + segment_diff;

                  if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

            if (overflow == No){
                  Ta = Tb; qa = qb; ++i;}}

      if (number_of_tries > 29){
            Ta=0;Tb=0;qb=0;return;}

      if (overflow == No){
            if (Tmax < Ttip)
                  if (Tb < Tbase ) Tmax = Ttip;

            overall_diff = (Tb - Tbase) / 2;

            if ( possible_overflow == Yes){
                        inc = inc / 2;
                        if (overall_diff > 0 )  Ttip = Ttip - inc;
                        else  Ttip = Ttip + inc;}
            else
                  Ttip = Ttip - overall_diff;
            number_of_tries++;
            if (Ttip < Tave) Ttip = Tave;}}
return;}
```

```
/* ***************************************************************************

      PROFILE AREA Analysis Program To Determine The Optimum Geometry

      Longitudinal Fin of Trapezoidal Profile

*************************************************************************** */

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "bessel.h"

#define      Yes    1
#define      No     0
#define      N      100
#define      sigma  5.66961e-8

double       max_length, length, depth, edge_width, base_width, Ta, Tb, qb=0, inc, Tave;
float        dq, V, qb_max, db, epsilon, taper_ratio, k, error=0.01, Tbase, K2, Heat, Ap;
int          i, Continue, number_of_passes, j, Exit, failures;
void         compute();

/* ************************************************************************

      MAIN PROGRAM

************************************************************************ */

int main(){

FILE *out;
randomize();
error = error / 100;
clrscr();

for (j=1; j < 102; j++){

      number_of_passes = 0;
      epsilon = (float) ( random(98) );
      epsilon = .01 + epsilon / 100;
      depth =  (float) (20 + random(200));
      depth = depth / 100;
      k = 10 + random(200);
      Tbase = 100 + random(500);
      Heat = (float) (25 + random(1000));
      Heat = Heat / depth;
      V = (float) ( random(80));
      V = .01 + V / 100;
      V = 0;
      taper_ratio = (float) (random(98));
      taper_ratio = 0.01 + taper_ratio / 100;

      Continue = Yes;
      Exit = No;
      qb_max = -1;
      dq = 0;
      qb = 0;
      Ap = (.63077 * taper_ratio + 1.31723) * pow(Heat,3) / ( k * pow(sigma,2) * pow(epsilon,2) *
                  pow(Tbase,9));
      length = .82 * (-.0315113 * taper_ratio + 1.0425113) * Heat / ( sigma * epsilon *
                  pow(Tbase,4));
      db = length / 20016;

      length = length - db;
      base_width = 2 * Ap / ( length * ( taper_ratio + 1));
      edge_width = taper_ratio * base_width;
```

```
        if (depth < 10*base_width) Continue = No;
        if (length < 10*base_width) Continue = No;

        while (Continue == Yes){

                length = length + db;
                base_width = 2 * Ap / ( length * ( taper_ratio + 1));
                edge_width = taper_ratio * base_width;

                failures = 0;
                number_of_passes++;

                compute();

                while (qb == 0 && Continue == Yes){
                        failures++;
                        length = length + db / 2;
                        base_width = 2 * Ap / ( length * ( taper_ratio + 1));
                        edge_width = taper_ratio * base_width;
                        compute();
                        if (failures == 3){
                                Continue = No;
                                number_of_passes = 1;}}

                dq = qb - dq;
                cprintf("\r\nwidth = %4.3f length = %4.6f heat = %4.6f  dq = %4.6f", base_width*1000,
                        length, qb, dq);
                dq = qb;

                if (qb_max < qb ){
                        max_length = length;
                        qb_max = qb;}
                else
                        Continue = No; }

        if (number_of_passes > 2){
                length = max_length;
                base_width = 2 * Ap / ( length * ( taper_ratio + 1));
                edge_width = taper_ratio * base_width;
                compute();
                cprintf("\r\nWriting Rec # %d d = %4.3f  b = %4.6f qb = %4.3f Tb = %4.1f Ap = %4.1f",
                        j, base_width, length, qb, Tbase, Ap);
                out = fopen("OPTTRAP.MAT","a");
                fprintf(out, "%d,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e\r\n", j,
                   depth, length, Ap, k, Tbase, epsilon, V, qb, taper_ratio, base_width, edge_width );
                fclose(out);}
        else{
                cprintf("\r\nIgnoring This Set Of Parameters");
                j=j-1;}}

return 0;}

void compute(void){

double          m, hr, n, ua, ub, A, B, C, D, fin_taper, I1UA, K0UB, I0UB, K1UA, K0UA, I0UA, K1UB;
double          qa, b1, dH, a, a_width, b_width, Ttip, temp, segment_diff, I1UB;
double          b, inc, Tav, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000;
int             overflow, possible_overflow = No, number_of_tries = 0;

b1 = length / ( 1 - taper_ratio );
fin_taper = atan2( base_width, 2 * b1 );
dH = length / N;

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
        Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
else
```

```
      Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      qa = sigma * edge_width * epsilon * pow(Ta, 4) * depth;
      a = b1 - length;

      while (i <= N && overflow == No){

            segment_diff = 10;
            a_width = edge_width * a / ( b1 - length);
            b_width = edge_width * ( a + dH ) / ( b1 - length);

            while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                              Tav = (Tb + Ta) / 2;
                              hr = sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav));
                              n =  hr / (k * sin(fin_taper));
                              ua =  2 * sqrt(a * n );
                              ub =  2 * sqrt( (a + dH) * n );
                              I1UA = I1(ua);
                              K0UB = K0(ub);
                              I0UB = I0(ub);
                              K1UA = K1(ua);
                              K0UA = K0(ua);
                              I0UA = I0(ua);
                              K1UB = K1(ub);
                              I1UB = I1(ub);
                              A = ua * (I1UA*K0UB + I0UB*K1UA);
                              B = ub*ub*(I0UB*K0UA - I0UA*K0UB) / ( 2*k*b_width*depth*n);
                              C = 2*k*a_width*depth*n*ub*(I1UB*K1UA - I1UA*K1UB)/(ua);
                              D = ub*( I0UA*K1UB + I1UB*K0UA);
                  temp = A * Ta + B * qa;
                  qb   = C * Ta + D * qa;
                  segment_diff = ( temp - Tb) / 2;
                  Tb = temp + segment_diff;

                  if (Tb > Tbase + 50 || Tb < 0){
                        overflow   = Yes; possible_overflow = Yes;
                        if (Ttip < Tbv_min ) Tbv_min = Ttip;
                        inc = (Tbv_min - Tmax) / 2;
                        Ttip = fabs(Tbv_min - inc);
                        if (Ttip < Tave) Ttip = Tave;}}

            if (overflow == No){
                  Ta = Tb; qa = qb; a = a + dH; ++i;}}

      if (number_of_tries > 29){
            Ta=0;Tb=0;qb=0;return;}

      if (overflow == No){
            if (Tmax < Ttip)
                  if (Tb < Tbase)  Tmax = Ttip;

            overall_diff = (Tb - Tbase) / 2;

            if ( possible_overflow == Yes){
                        inc = inc / 2;
                        if (overall_diff > 0 )  Ttip = Ttip - inc;
                        else  Ttip = Ttip + inc;}
            else
                  Ttip = Ttip - overall_diff;
            number_of_tries++;
            if (Ttip < Tave) Ttip = Tave;}}

return;}
```

```
/* ****************************************************************************

       PROFILE AREA Analysis Program Tb Determine The Optimum Geometry

       Longitudinal Fin of Triangular Profile

****************************************************************************** */

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "bessel.h"

#define      Yes    1
#define      No     0
#define      N      99
#define      sigma  5.66961e-8

double       length, depth, edge_width, base_width, max_length;
double       Ta, Tb, qb=0;
float        V, qb_max, db, epsilon, k, error=0.01, taper_ratio, Tbase, K2, Heat, Ap;
int          i, Exit, Continue, number_of_passes, j, failures;
void         compute();

/* **************************************************************************

       MAIN PROGRAM

**************************************************************************** */

int main(){
float dq;

FILE *out;
randomize();
error = error / 100;
clrscr();

for (j=1; j < 102; j++){

       number_of_passes = 0;
       epsilon = (float) ( random(98) );
       epsilon = .01 + epsilon / 100;
       depth =  (float) (20 + random(200));
       depth = depth / 100;
       k = 10 + random(200);
       Tbase = 100 + random(500);
       Heat = (float) (25 + random(1000));
       Heat = Heat / depth;
       V = (float) ( random(80));
       V = .01 + V / 100;
       V = 0;
       taper_ratio = (float) (random(98));
       taper_ratio = 0.01 + taper_ratio / 100;
       taper_ratio = 0;

       Exit = No;
       Continue = Yes;
       qb_max = -1;
       qb = 0;
       dq = 0;
       Ap = 1.304 * pow(Heat,3) / ( k * pow(sigma,2) * pow(epsilon,2) * pow(Tbase,9));
       length = .95 * Heat / ( sigma * epsilon * pow(Tbase,4));
       db = length / 20016;
       length = length - db;
       base_width = 2 * Ap / length;
```

```
        if (depth < 10*base_width) Continue = No;
        if (length < 10*base_width) Continue = No;

        while (Continue == Yes){

                length = length + db;
                base_width = 2 * Ap / length;

                number_of_passes++;
                failures = 0;

                compute();

                while (qb == 0 && Continue == Yes){
                        failures++;
                        length = length + db / 2;
                        base_width = 2 * Ap / length;
                        compute();
                        if (failures == 3){
                                Continue = No;
                                number_of_passes = 1;}}

                dq = qb - dq;
                cprintf("\r\ndelta = %4.3f length = %4.6f heat = %4.6f  dq = %4.6f", base_width*1000,
                        length, qb, dq);
                dq = qb;

                if (qb_max < qb ){
                        max_length = length;
                        qb_max = qb;}
                else
                        Continue = No; }

        if (number_of_passes > 2){
                length = max_length;
                base_width = 2 * Ap / length;
                compute();
                cprintf("\r\nWriting Rec # %d d = %4.3f  b = %4.6f qb = %4.3f Tb = %4.1f Ap = %4.1f",
                        j, base_width, length, qb, Tbase, Ap);
                out = fopen("OPTTRI.MAT","a");
                edge_width = 0;
                fprintf(out, "%d,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e,  %e\r\n", j,
                   depth, length, Ap, k, Tbase, epsilon, V, qb, taper_ratio, base_width, edge_width );
                fclose(out);}
        else{
                cprintf("\r\nIgnoring This Set Of Parameters");
                j=j-1;}}

return 0;}

void compute(void){

double          m, hr, A, B, C, D, I1UA, K0UB, I0UB, K1UA, K0UA, I0UA, K1UB, I1UB;
double          fin_taper, b1, db, a, ua, ub, a_width, b_width;
double          inc, Tave, overall_diff = 100, Tmax = 0, Tbv_min = 100000;
double          Tav, Ttip, temp, qa, segment_diff;
int             overflow, possible_overflow = No, number_of_tries = 0;

edge_width = base_width / 100;
db = length / ( N + 1 );
b1 = length;
length = length - db;
fin_taper = atan2( base_width, 2*b1 );

K2 = 2 * epsilon * sigma * V * pow(Tbase,4);
Tave = Tbase * pow(V, 0.25) + 0.0001;

if (V==0)
        Ttip = .8625 * Tbase * sqrt( 2 * V ) + ( ( Tbase - (Tbase * sqrt( 2 * V )))/2);
```

```
else
      Ttip = Tave + 0.0001;

inc = .5 * Ttip; i=1;

while (fabs(overall_diff) > ( Tbase * error)  || i <= N ){

      overflow = No; i = 1; Ta = Ttip; Tb = Ta;
      segment_diff = 10;

      while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                              Tav = (Tb + Ta) / 2;
                              hr = sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav));
                              m =  hr / (k * sin(fin_taper));
                              ub =  2 * sqrt( db * m );
                              I0UB = I0(ub);
                              K1UB = K1(ub);
                              I1UB = I1(ub);
            C = 2*k*edge_width* depth*pow(m,2)*I1UB/(ub * I0UB);


            temp = Ta / I0UB;
            qb   = C * temp;
            segment_diff = ( temp - Tb) / 2;
            Tb = temp + segment_diff;

            if (Tb > Tbase + 50 || Tb < 0){
                  overflow   = Yes; possible_overflow = Yes;
                  if (Ttip < Tbv_min ) Tbv_min = Ttip;
                  inc = (Tbv_min - Tmax) / 2;
                  Ttip = fabs(Tbv_min - inc);
                  if (Ttip < Tave) Ttip = Tave;}}

      if (overflow == No){
            Ta = Tb; qa = qb; ++i;}

      a = b1 - length;

      while (i <= N && overflow == No){

            segment_diff = 10;
            a_width = edge_width * a / ( b1 - length);
            b_width = edge_width * ( a + db ) / ( b1 - length);

            while (fabs(segment_diff) > ( Tb * error) / N && overflow == No){

                              Tav = (Tb + Ta) / 2;
                              hr = sigma * epsilon * pow(Tav, 3) - (K2 / (2 * Tav));
                              m =  hr / (k * sin(fin_taper));
                              ua =  2 * sqrt(a * m );
                              ub =  2 * sqrt( (a + db) * m );
                              I1UA = I1(ua);
                              K0UB = K0(ub);
                              I0UB = I0(ub);
                              K1UA = K1(ua);
                              K0UA = K0(ua);
                              I0UA = I0(ua);
                              K1UB = K1(ub);
                              I1UB = I1(ub);
                              A = ua * (I1UA*K0UB + I0UB*K1UA);
                              B = ub*ub*(I0UB*K0UA - I0UA*K0UB) / ( 2*k*b_width*depth*m);
                              C = 2*k*a_width*depth*m*ub*(I1UB*K1UA - I1UA*K1UB)/(ua);
                              D = ub*( I0UA*K1UB + I1UB*K0UA);

                  temp = A * Ta + B * qa;
                  qb   = C * Ta + D * qa;
                  segment_diff = ( temp - Tb) / 2;
                  Tb = temp + segment_diff;
```

```
                    if (Tb > Tbase + 50 || Tb < 0){
                            overflow   = Yes; possible_overflow = Yes;
                            if (Ttip < Tbv_min ) Tbv_min = Ttip;
                            inc = (Tbv_min - Tmax) / 2;
                            Ttip = fabs(Tbv_min - inc);
                            if (Ttip < Tave) Ttip = Tave;})

            if (overflow == No){
                    Ta = Tb; qa = qb; a = a + db; ++i;})

    if (number_of_tries > 29){
            Ta=0; Tb=0; qb=0; return;}

    if (overflow == No){
            if (Tmax < Ttip)
                    if (Tb < Tbase) Tmax = Ttip;

            overall_diff = (Tb - Tbase) / 2;

            if ( possible_overflow == Yes){
                            inc = inc / 2;
                            if (overall_diff > 0 )  Ttip = Ttip - inc;
                            else  Ttip = Ttip + inc;}
            else
                    Ttip = Ttip - overall_diff;
            number_of_tries++;
            if (Ttip < Tave) Ttip = Tave;})

length = length + db;
return;}
```

APPENDIX C MATLAB EVALUATION M FILES

```
/* ******************************************************************************

      MATLAB EVALUATION M FILE To Determine The Optimum Geometry

      Longitudinal Fin of Rectangular Profile

****************************************************************************** */

load rect;
D = rect;

N = 100;
sigma = 5.66961e-8;

L  = D(:,2);
b  = D(:,3);
A  = D(:,4);
k  = D(:,5);
To = D(:,6);
e  = D(:,7);
V  = D(:,8);
qo = D(:,9);
TR = D(:,10);
db = D(:,11);
de = D(:,12);
I  = [1:101];

C3 = A.* k.* (((sigma*e).^2).* To.^9).* (L./qo).^3;
C1 = b.* sigma.* e.* (To.^4).* (L./qo);
C2 = db.* k.* sigma.* e.* (To.^5).* (L./qo).^2;

disp('RECT V = 0')
disp(' ')
disp('Constant One:  Polynomial p is:')
p = polyfit(TR, C1, 1)
fit1 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C1-fit1).^2)/(N-1))
pause

disp(' ')
disp('Constant Two:  Polynomial p is:')
p = polyfit(TR, C2, 1)
fit2 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C2-fit2).^2)/(N-1))
pause

disp(' ')
disp('Constant Three:  Polynomial p is:')
p = polyfit(TR, C3, 1)
fit3 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C3-fit3).^2)/(N-1))
```

```
/* ***************************************************************************

      MATLAB EVALUATION M FILE To Determine The Optimum Geometry

      Longitudinal Fin of Trapezoidal Profile

*************************************************************************** */

load trap;
D = trap;

N = 100;
sigma = 5.66961e-8;

L  = D(:,2);
b  = D(:,3);
A  = D(:,4);
k  = D(:,5);
To = D(:,6);
e  = D(:,7);
V  = D(:,8);
qo = D(:,9);
TR = D(:,10);
db = D(:,11);
de = D(:,12);
I  = [1:101];

C3 = A.* k.* (((sigma*e).^2).* To.^9).* (L./qo).^3;
C1 = b.* sigma.* e.* (To.^4).* (L./qo);
C2 = db.* k.* sigma.* e.* (To.^5).* (L./qo).^2;

disp('TRAP V = 0')
disp(' ')
disp('Constant One:  Polynominal p is:')
p = polyfit(TR, C1, 1)
fit1 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C1-fit1).^2)/(N-1))
pause

disp(' ')
disp('Constant Two:  Polynominal p is:')
p = polyfit(TR, C2, 2)
fit2 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C2-fit2).^2)/(N-1))
pause

disp(' ')
disp('Constant Three:  Polynominal p is:')
p = polyfit(TR, C3, 1)
fit3 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C3-fit3).^2)/(N-1))
pause
```

```
/* ****************************************************************************

      MATLAB EVALUATION M FILE To Determine The Optimum Geometry

      Longitudinal Fin of Triangular Profile

**************************************************************************** */

load tri;
D = tri;

N = 100;
sigma = 5.66961e-8;

L  = D(:,2);
b  = D(:,3);
A  = D(:,4);
k  = D(:,5);
To = D(:,6);
e  = D(:,7);
V  = D(:,8);
qo = D(:,9);
TR = D(:,10);
db = D(:,11);
de = D(:,12);
I  = [1:101];

C3 = A.* k.* (((sigma*e).^2).* To.^9).* (L./qo).^3;
C1 = b.* sigma.* e.* (To.^4).* (L./qo);
C2 = db.* k.* sigma.* e.* (To.^5).* (L./qo).^2;

disp('TRI V = 0')
disp(' ')
disp('Constant One:  Polynominal p is:')
p = polyfit(TR, C1, 1)
fit1 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C1-fit1).^2)/(N-1))
pause

disp(' ')
disp('Constant Two:  Polynominal p is:')
p = polyfit(TR, C2, 1)
fit2 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C2-fit2).^2)/(N-1))
pause

disp(' ')
disp('Constant Three:  Polynominal p is:')
p = polyfit(TR, C3, 1)
fit3 = polyval(p,TR);
disp('Mean Square Error:')
std_err = sqrt(sum((C3-fit3).^2)/(N-1))
```

LIST OF REFERENCES

1. Murray, W.M., "Heat Transfer Through An Annular Disk or Fin of Uniform Thickness," *ASME Journal of Applied Mechanics,* v. 60, p. A78.

2. Gardner, K.A., "Efficiency of Extended Surface," *Transactions ASME,* v.67, p. 621.

3. Kraus, Allan D., *Analysis and Evaluation of Extended Surface Thermal Systems,* p. 87, Hemisphere Publishing Company, 1982.

4. Kraus, Allan D., *Analysis and Evaluation of Extended Surface Thermal Systems,* p. 3-4, Hemisphere Publishing Company, 1982.

5. Kraus, Allan D., *Analysis and Evaluation of Extended Surface Thermal Systems,* p. 92, Hemisphere Publishing Company, 1982.

6. Kraus, Allan D., *Analysis and Evaluation of Extended Surface Thermal Systems,* p. 101, Hemisphere Publishing Company, 1982.

7. Kraus, Allan D., *Analysis and Evaluation of Extended Surface Thermal Systems,* p. 200, Hemisphere Publishing Company, 1982.

8. Gieck, Kurt, *Engineering Formulas 4th Edition,* pp. Z1-Z4, McGraw-Hill Book Company, Inc., 1983.

9. Kern, Donald Q., and Kraus, Allan D., *Extended Surface Heat Transfer,* p. 237, McGraw-Hill Book Company, Inc., 1972.

10. Chung, B.T.F., and Nguyen, L.D., "AIAA-86-0150 Optimization of Design Parameters For Radiating Longitudinal Fins of Various Geometries," presented at the 24th AIAA Aerospace Sciences Meeting, p. 3, Reno, NV, January 1986.

11. Chung, B.T.F., and Nguyen, L.D., "AIAA-86-0150 Optimization of Design Parameters For Radiating Longitudinal Fins of Various Geometries," presented at the 24th AIAA Aerospace Sciences Meeting, p. 3, Reno, NV, January 1986.

12. Liu, C.Y., "On Minimum Weight Rectangular Radiating Fins," *Journal of Aerospace Science,* v. 27, p. 871, 1960.

13. Bartas, J.G., and Sellers, W.H., "Radiation Fin Effectiveness," *Journal of Heat Transfer,* v. 82, p. 73, February 1960.

14. Nilson, N., and Curry, R., "The Minimum Weight Straight Fin of Triangular Profile Radiating to Space," *Journal of Aerospace Science,* v. 27, p. 146, 1960.

15. Kern, Donald Q., and Kraus, Allan D., *Extended Surface Heat Transfer,* p. 237, McGraw-Hill Book Company, Inc., 1972.

16. Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William, *Numerical Recipes The Art of Scientific Computing,* pp. 176-80, Cambridge University Press, 1986.

BIBLIOGRAPHY

Aparecido, J.B., and Cotta, R.M., "Improved One-Dimensional Fin Solutions," *Heat Transfer Engineering*, v. 11, pp. 49-59, January 1990.

Boyce, William, and Diprima, Richard C., *Elementary Differential Equations and Boundary Value Problems*, John Wiley and Sons, Inc., 1986.

Callinan, Joseph P., and Berggren, Willard P., "Some Radiator Design Criteria for Space Vehicles," *Journal of Heat Transfer*, v. 81, pp. 237-244.

Haberman, Richard, *Elementary Applied Partial Differential Equations,* Prentice-Hall, Inc., 1987.

Holman, J.P., *Experimental Methods For Engineers*, McGraw-Hill Book Company, 1979.

Janna, William S., *Engineering Heat Transfer*, Prindle, Weber and Schmidt Publishers, 1986.

National Aeronautics and Space Administration Technical Note D-2168, *Heat Rejection and Weight Characteristics of Fin-Tube Space Radiators With Tapered Fins*, by Haller, Henry C., Wesling, Gordon C., and Lieblein, Seymour, February 1964.

Schnurr, N.M., Shapiro, A.B., and Townsend, M.A., "Optimization of Radiation Fin Arrays With Respect to Weight," *Journal of Heat Transfer*, v. 98, pp. 643-648, November 1976.

Snider, A.D., and Kraus, A.D., "A General Extended Surface Analysis Method," *Journal of Heat Transfer*, v. 103, pp. 699-704, November 1981.

Snider, A.D., and Kraus, A.D., "Recent Developments in the Analysis and Design of Extended Surface," *Journal of Heat Transfer*, v. 105, pp. 302-306, May 1983.

Turbo C++, Borland International, Inc., 1990.

Van Valkenburg, M.E., *Network Analysis*, Prentice-Hall, Inc., 1974.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5100	1
4.	Prof. Allan D. Kraus, Code EC/Ks Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5100	3
5.	Prof. Rudolf Panholzer, Code EC/Pz Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5100	1
6.	Prof. Yogendra K. Joshi, Code ME/Ji Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5100	1
7.	Mrs Evelyn Goble 510 Swing Avenue Findlay, Ohio 45840	2

8.	Milo Smith, Jr. 5325 South West 15th Street Topeka, Kansas 66604	1
9.	LT Steven M. Smith Mare Island Naval Shipyard Vallejo, California 94592-5100	2